

目录

目录.....	1
1 SDK121X 学习评估板简介	4
1.1 SDK121X 学习评估板概述	4
1.2 SDK121X 学习评估板硬件资源汇总	4
1.3 SDK121X 学习评估板软件资源汇总	6
2 赛元 MCU 软硬件开发平台介绍.....	8
2.1 开发平台 KEIL C.....	8
2.2 烧录仿真工具.....	8
2.3 PC 端烧录软件 SOC PRO51	8
3 学习评估板各模块电路图	9
3.1 主控 MCU 及外部晶振.....	9
3.2 电源模块.....	12
3.3 LED/RGB/LCD/数码管显示单元	12
3.4 按键与蜂鸣器.....	13
3.5 UART0 与 Timer2	13
3.6 三选一串行接口 USCI	14
3.7 ADC 与 CMP	14
4.SDK121X 学习评估板示例	15
4.1 GPIO 示例.....	15
4.1.1 SDK1211/1212GPIO 示例	15
4.1.2 SDK1213 GPIO 示例	16
4.2 外部中断示例.....	17
4.2.1 SDK1211/1212 INT 示例.....	17
4.2.2 SDK1213 INT 示例.....	17
4.3 PWM 示例.....	19
4.3.1 SDK1211 /1212 PWM 示例	19
4.3.2 SDK1213 PWM 示例.....	20

4.4 Timer0 示例	22
4.4.1 SDK1211 /1212 Timer0 示例.....	22
4.4.2 SDK1213 Timer0 示例	23
4.5 Timer1 示例	24
4.5.1 SDK1211 /1212 Timer1 示例.....	24
4.5.2 SDK1213Timer1 示例	25
4.6 Timer2 示例	28
4.6.1 SDK1211 /1212 Timer2 示例.....	28
4.6.2 SDK1213 Timer2 示例	29
4.7LED 显示驱动示例	32
4.8 LCD 显示驱动示例	33
4.9 模拟比较器示例	34
4.9.1 SDK1211 /1212 模拟比较器示例.....	34
4.10 低频时钟定时器 BTM 示例	35
4.10.1 SDK1211/1212 BTM 示例	35
4.10.2 SDK1213 BTM 示例.....	36
4.11 ADC 示例	37
4.11.1 SDK1211/1212 ADC 示例	37
4.11.2 SDK1213 ADC 示例.....	38
4.12 串口 UART0 示例	40
4.12.1 SDK1211/1212 UART0 示例	40
4.12.2 SDK1213 UART0 示例.....	41
4.13 IAP 示例	42
4.14 乘除法器示例	43
4.14.1 SDK1211/1212 MDU 示例	43
4.14.2 SDK1213 MDU 示例	43
4.15 USCI1_UART 示例	45
4.15.1 SDK1211/1212 UART1 示例	45
4.15.2 SDK1213 USCI1_UART 示例.....	45
4.16 SPI1 示例	47
4.16.1 SDK1211/1212 SPI1 示例.....	47
4.16.2 SDK1213 SPI1 示例.....	48
4.17 触控按键电路 Touch Key 示例	49
4.17.1 SDK1211/1212 TK 示例.....	49
4.17.2 SDK1213 TK 示例.....	50
4.18 功能复用示例	52
4.18.1 SDK1211/1212 功能复用示例.....	52
4.18.1 SDK1213 功能复用示例.....	53

4.19 规格更改记录	55
--------------------------	-----------

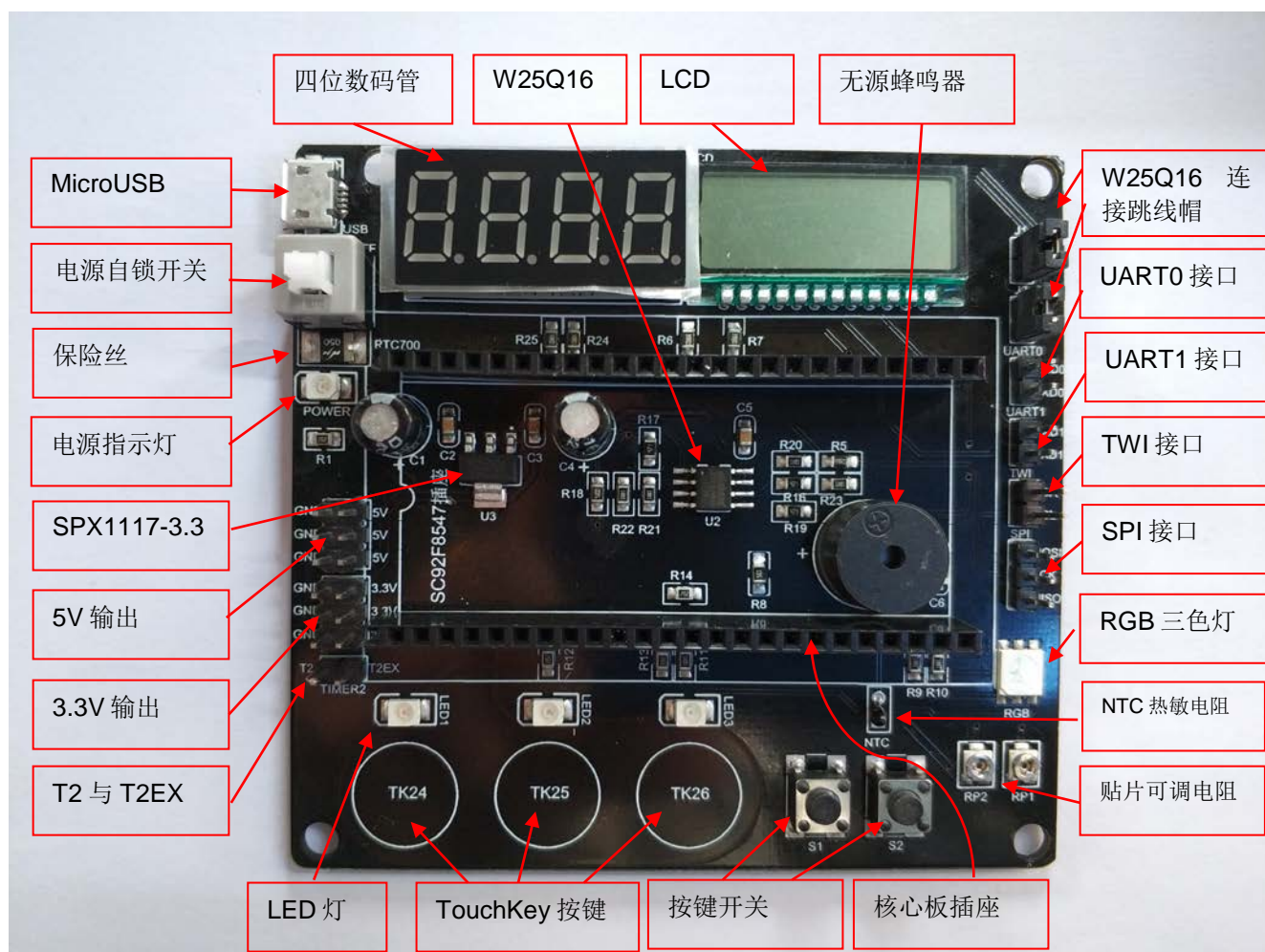
1 SDK121X 学习评估板简介

1.1 SDK121X 学习评估板概述

赛元 SDK121X 学习评估板，由学习评估板 SDK1010 以及核心板 SDK1211/ 1212/1213 组成。本学习评估板配备常用的单片机外围资源，自带调试下载接口，配合学习评估板提供的示例程序(SDK1213 示例程序会用易码魔盒讲解)，可以让用户在最短的时间，熟悉并掌握赛元 MCU 相关的编程方法。本学习评估板非常适合初步接触赛元 MCU 的用户自学使用。

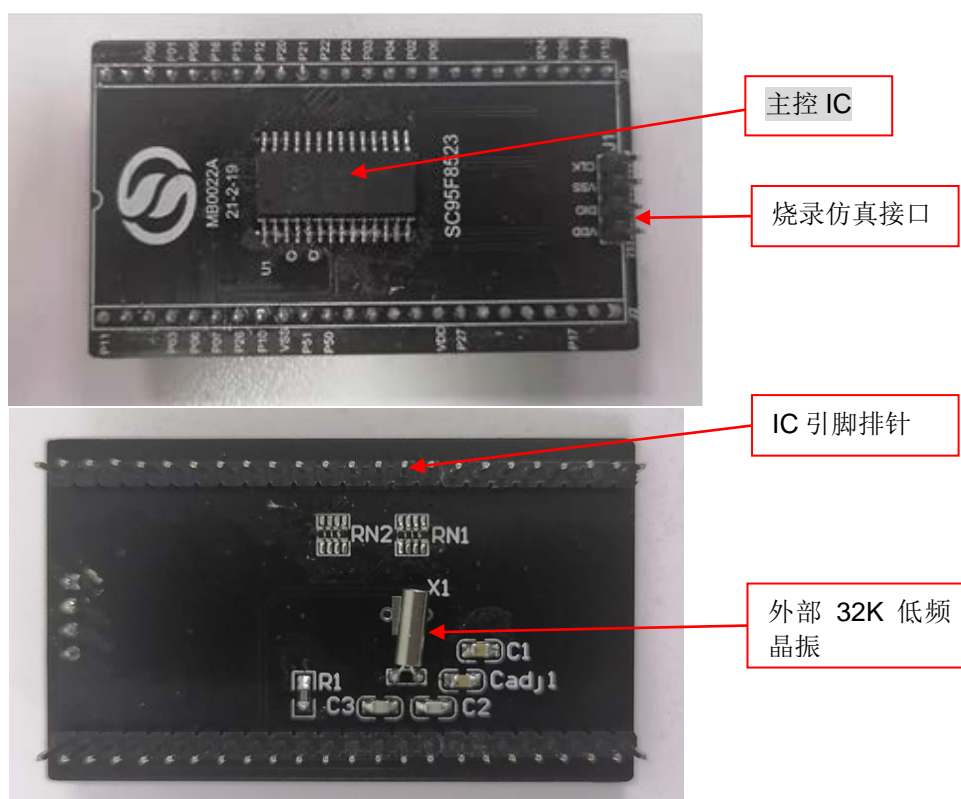
1.2 SDK121X 学习评估板硬件资源汇总

下图为学习评估板 SDK1010 资源简图。



底板 SDK1010

学习评估板核心板资源简图以 SDK1213 为例，如下，上部分为正面，下部分为反面。底板与核心板拼接时，核心板赛元 logo 一端应对准底板丝印下凹一端。



核心板 SDK1213

SDK1211 核心板板载资源如下：

- CPU: SC95F8617, 工作电压为 2.0V~5.0V, ROM 为 64KB, RAM 为 4KB, 系统时钟可选 32/16/4MHz
- 一组烧录仿真引脚
- 一个外接 32K Hz 低频晶振

SDK1212 核心板板载资源如下：

- CPU: SC95F8616, 工作电压为 2.0V~5.0V, ROM 为 64KB, RAM 为 4KB, 系统时钟可选 32/16/4MHz
- 一组烧录仿真引脚
- 一个外接 32K Hz 低频晶振

SDK1213 核心板板载资源如下：

- CPU: SC95F8523, 工作电压为 2.0V~5.5V, ROM 为 32KB, RAM 为 4KB, 系统时钟可选 32/16/4MHz
- 一组烧录仿真引脚
- 一个外接 32K Hz 低频晶振

SDK121X 学习评估板底板 SDK1010 板载资源如下：

- 一个 MicroUSB 外部供电接口
- 一个电源自锁开关, 使用外部供电时控制整个板子电源
- 一个保险丝, 防止芯片烧坏
- 一个电源指示灯, 红光
- 一个 SPX1117-3.3 芯片, 提供 3.3V 的稳压电源
- 一组 5V 电源供应口
- 一组 3.3V 电源供应口
- 一个无源蜂鸣器
- 一个四位共阴数码管
- 一个四位段码 LCD, 3.3V, 1/4Duty, 1/3Bias
- 一个 W25Q16 存储芯片, 16M 存储空间, 2.7~3.6V
- 一个 RGB 三色灯

- 一个 NTC 热敏电阻
- 两个按键开关
- 两个贴片可调电阻
- 三个 LED 灯，红光
- 三个 TouchKey 按键

以上板载资源，搭配主控 IC 自带硬件资源，足够用户熟悉并掌握赛元 MCU 用法。

1.3 SDK121X 学习评估板软件资源汇总

为了方便用户尽快熟悉赛元 MCU，在结合学习评估板资源的基础上，我们提供了 MCU 相关的例程代码，用户可通过烧录接口烧录 hex 文件，可直接在学习评估板上观察到对应现象。

学习评估板的例程列表如下表所示：

例程	描述	
GPIO	SDK1211	通过配置 GPIO 来操作 LED 灯，现象为 LED1 亮灭一段时间
	SDK1212	
	SDK1213	
INT	SDK1211	通过将 P04 配置为外部中断，下降沿触发。现象为当按下开关按键 S2 时，LED1 灯的亮灭状态翻转
	SDK1212	
	SDK1213	通过将 P07 配置为外部中断，下降沿触发。现象为当按下开关按键 S2 时，LED1 灯的亮灭状态翻转
PWM	SDK1211	通过不断循环改变 PWM40、PWM41、PWM53 的占空比，实现 RGB 三色灯的循环变色
	SDK1212	
	SDK1213	通过改变 PWM6 占空比，实现 LED1 呼吸灯效果
Timer0	SDK1211	使用 Timer0 定时功能，使 LED1 灯每隔 600ms 亮灭一次
	SDK1212	
	SDK1213	
Timer1	SDK1211	将 Timer1 设为计数器，计数值为 1。现象为当开关按键 S1 按下时，LED1 灯的亮灭状态改变一次。由于按键抖动，S1 按下一次 LED1 灯状态可能改变多次
	SDK1212	
	SDK1213	将 Timer1 设为计数器，P12 输出 1HZ 的方波，（短接 UART0 的 RX 和 TX）数码管显示计数值
Timer2	SDK1211	将 Timer2 设为捕获模式，将 Timer0 在 P12（SCK）口产生的方波接在 T2EX 引脚上，LED 数码管显示该方波的周期值，单位为微秒
	SDK1212	
	SDK1213	将 Timer2 设为捕获模式，Timer0 控制 P0.7 输出周期 20ms 的方波，将 Timer2 短接，数码管上显示该方波的周期，单位毫秒
DDIC_LED	SDK1211	启动 LED 硬件驱动电路，使 4 位数码管显示的值每秒加 1。（学习评估板 LCD 与数码管 SEG 口相同，故使用 LED 时 LCD 会显示乱码，正常现象）
	SDK1212	
DDIC_LCD	SDK1211	启动 LCD 硬件驱动电路，使 4 位 LCD 显示屏显示的值每秒加 1
	SDK1212	
		-
ACMP	SDK1211	模拟比较器功能，通过调节 RP1 电阻可控制比较器正端输入电压（P43 口电压），调节 RP2 电阻可控制比较器负端比较电压（P44 口电压），当正端电压大于负端电压时，LED2 灯亮，反之 LED3 灯亮
	SDK1212	
		-
BTM	SDK1211	低频时钟定时器，使用外部晶振，需勾选 option 项，此时有且只有 LED1 灯以 0.25s 周期闪烁，其余灯灭
	SDK1212	
	SDK1213	
ADC	SDK1211	ADC 采集 NTC 热敏电阻电压，转换为当前室温，数码管显示室温
	SDK1212	
	SDK1213	

IAP	SDK1211	通过 IAP 将数组的值写进 ROM，再读出 ROM 的值与数组的值比较，若相同，则 UART0 发送 ROM START 与 END 至上位机，若不相同，上位机将显示详细信息
	SDK1212	
	SDK1213	
MDU	SDK1211	乘除法器，若乘法结果正确，则 LED1 灯亮，若除法结果正确，则 LED2 灯亮
	SDK1212	
	SDK1213	乘除法器，若乘除法结果正确，则 LED1 闪烁两次
UART0	SDK1211	UART0 与上位机通信，波特率 9600，上位机显示“UART0 is OK！”
	SDK1212	
	SDK1213	
UART1	SDK1211	UART1 与上位机通信，波特率 9600，上位机发送数据到 MCU，MCU 将所接收到的内容发出
	SDK1212	
	SDK1213	
SPI	SDK1211	通过 SPI 操作 W25Q16，将数组数据 0~9 写入 W25Q16，再将 W25Q16 中的值读出来并通过 UART0 发送给上位机，上位机显示 0~9
	SDK1212	
	SDK1213	
TWI	SDK1211	与 IIC 协议相同，可做主从机，学习评估板无相关硬件，只提供程序配置
	SDK1212	
	SDK1213	
TK	SDK1211	按下 TouchKey 按键，上方对应的 LED 灯亮，蜂鸣器鸣响；
	SDK1212	
	SDK1213	按下 TouchKey 按键，上方对应的 LED 灯亮
功能复用程序	SDK1211	将多数软件资源整合，用户可通过按键选择所要观察的现象。 SC95F8617/SC95F8616 通过 TK24/TK25/TK26 按键进行模式选择，S1 进入当前模式，S2 退出当前模式； SC95F8617/95F8616 通过 S1 进行模式选择，S2 确认进入/退出当前模式
	SDK1212	
	SDK1213	将多数软件资源整合，用户可通过按键选择所要观察的现象。 1. SC95F8523 通过 S1 按键进行模式选择，TK24 进入/退出当前模式。 2. SC95F7523 通过 S1 按键单击进行模式选择，S1 按键长按进入/退出当前模式 Mode0: ADC 采集 NTC 热敏电阻电压，转换为当前室温，数码管显示室温。 Mode1: SC95F8523:将 Timer2 设为捕获模式，Timer4 控制 P0.7 输出周期 20ms 的方波，将 Timer2 短接，数码管上显示该方波的周期，单位毫秒。 SC95F7523: 将 Timer2 设为捕获模式，Timer4 控制 P1.2/P1.3 输出周期 20ms 的方波，将 Timer2 T2EX 短接，数码管上显示该方波的周期，单位毫秒。(注意 T2 和 S2 按键共用引脚) Mode2: UART0 与上位机通信，波特率 9600，上位机发送数据到 MCU，MCU 将所接收到的内容发出。 Mode3: 通过 SPI 操作 W25Q16，将数组数据 0~9 写入 W25Q16，再将 W25Q16 中的值读出来并通过 UART0 发送给上位机，上位机显示 0~9 Mode4: 通过改变 PWM6 占空比，实现 LED1 呼吸灯效果。

2 赛元 MCU 软硬件开发平台介绍

2.1 开发平台 KEIL C

赛元 MCU，采用 Keil C 即 KEIL uVISION 平台来开发，支持汇编语言以及 C 语言编写。

KEIL uVISION，是众多单片机应用开发软件中最优秀的软件之一，它支持众多不同公司的 MCS51 架构的芯片甚至 ARM，它集编辑，编译，仿真等于一体，界面与常用的微软 VC++ 界面相似，界面简洁，易学易用，在调试程序，软件仿真方面也有很强大的功能。

有关 KEIL C 的使用，请参考赛元官网资料[“赛元 MCU 工具使用说明”](#)文档的第二部分“赛元 MCU 的开发平台——Keil C”，有 Keil C 的安装及新建工程等使用说明。

2.2 烧录仿真工具

赛元目前使用的烧录工具有 SC-LINK，DPT52，PRO52。烧录工具使用前请安装赛元仿真插件。SC-LINK 适用于赛元 92F/93/95F 系列 IC 的脱机烧录、在线烧写、仿真以及 92F/93F/95F 系列触控 IC 的 Touch 调试。在线工具 DPT52 适用于赛元所有系列 IC 的在线编程、触控系列 IC 的调试以及部分系列 IC 的仿真。量产编程工具 PRO52 适用于赛元所有系列 IC 的量产烧写。有关赛元烧录仿真工具的使用与仿真插件的安装，请参考赛元官网资料[“赛元烧录仿真工具 SC LINK 使用说明”](#)、[“赛元在线开发工具 DPT52 使用说明”](#)、[“赛元量产编程工具 PRO52 使用说明”](#)。

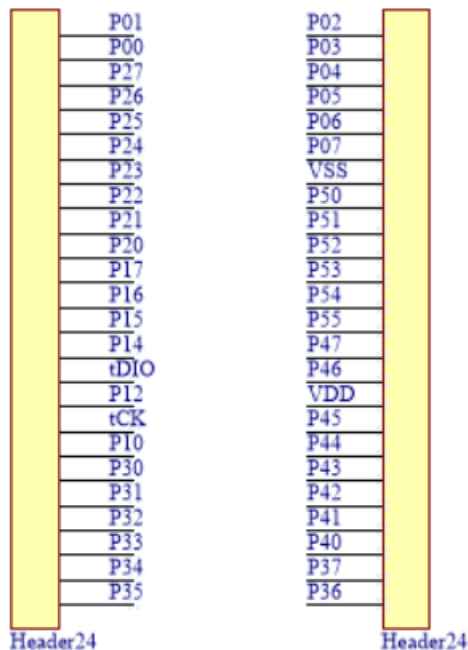
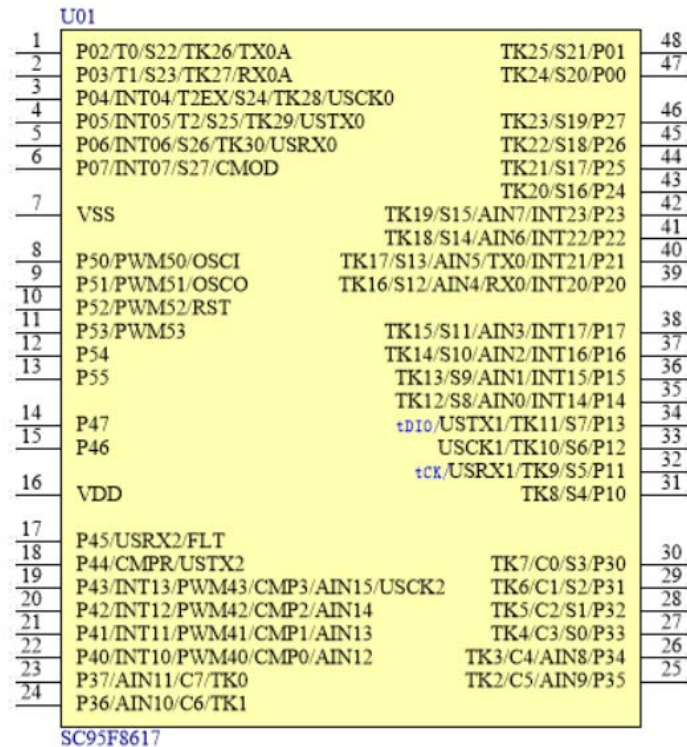
2.3 PC 端烧录软件 SOC PRO51

赛元 PC 端烧录上位机 SOC PRO51，支持 SC-LINK、DPT52、PRO52 等全系列烧录仿真工具。关于 SOC PRO51 的安装步骤与使用说明请参考赛元官网资料[“赛元 MCU 工具使用说明”](#)文档“1.3 软件安装步骤”与“1.4 开发工具功能说明与操作流程”（文档中烧录软件界面为旧版 V3.05 界面）。

3 学习评估板各模块电路图

3.1 主控 MCU 及外部晶振

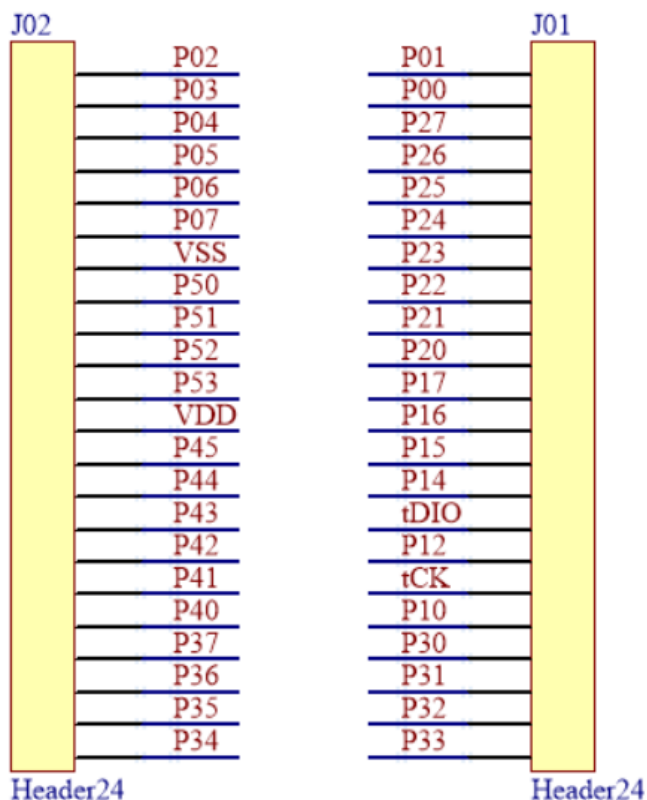
SDK1211 核心板使用 SC95F8617 作为主控芯片。同时外接 32K Hz 低频晶振，可作为 BTM 低频时钟定时器振荡器。



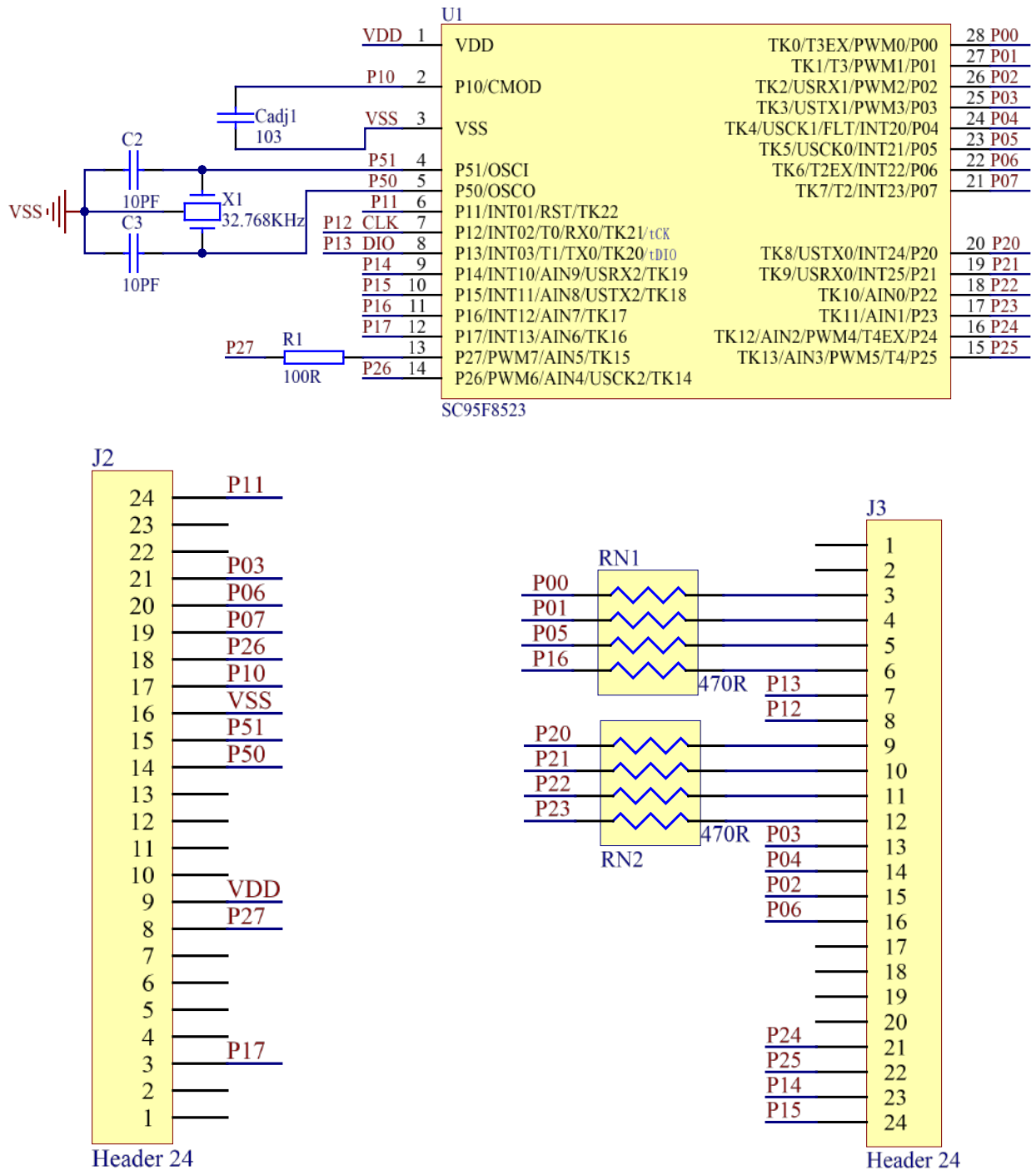
SDK1212 核心板使用 SC95F8616 作为主控芯片。同时外接 32KHz 低频晶振，可作为 BTM 低频时钟定时器振荡器。

U2			
1	P02/T0/S22/TK26/TX0A	TK25/S21/P01	44
2	P03/T1/S23/TK27/RX0A	TK24/S20/P00	43
3	P04/INT04/T2EX/S24/TK28/USCK0		
4	P05/INT05/T2/S25/TK29/USTX0		
5	P06/INT06/S26/TK30/USRX0	TK23/S19/P27	42
6	P07/INT07/S27/CMOD	TK22/S18/P26	41
		TK21/S17/P25	40
7	VSS	TK20/S16/P24	39
		TK19/S15/AIN7/INT23/P23	38
8	P50/PWM50/OSCI	TK18/S14/AIN6/INT22/P22	37
9	P51/PWM51/OSCO		
10	P52/PWM52/RST	TK17/S13/AIN5/TX0/INT21/P21	36
11	P53/PWM53	TK16/S12/AIN4/RX0/INT20/P20	35
		TK15/S11/AIN3/INT17/P17	34
12	VDD	TK14/S10/AIN2/INT16/P16	33
		TK13/S9/AIN1/INT15/P15	32
13	P45/USRX2/FLT	TK12/S8/AIN0/INT14/P14	31
14	P44/CMPR/USTX2	tDIO/STX1/TK11/S7/P13	30
15	P43/INT13/PWM43/CMP3/AIN15/USCK2	USCK1/TK10/S6/P12	29
16	P42/INT12/PWM42/CMP2/AIN14	tCK/SRX1/TK9/S5/P11	28
17	P41/INT11/PWM41/CMP1/AIN13	TK8/S4/P10	27
18	P40/INT10/PWM40/CMP0/AIN12		
19	P37/AIN11/C7/TK0	TK7/C0/S3/P30	26
20	P36/AIN10/C6/TK1	TK6/C1/S2/P31	25
21	P35/AIN9/C5/TK2	TK5/C2/S1/P32	24
22	P34/AIN8/C4/TK3	TK4/C3/S0/P33	23

SC95F8616

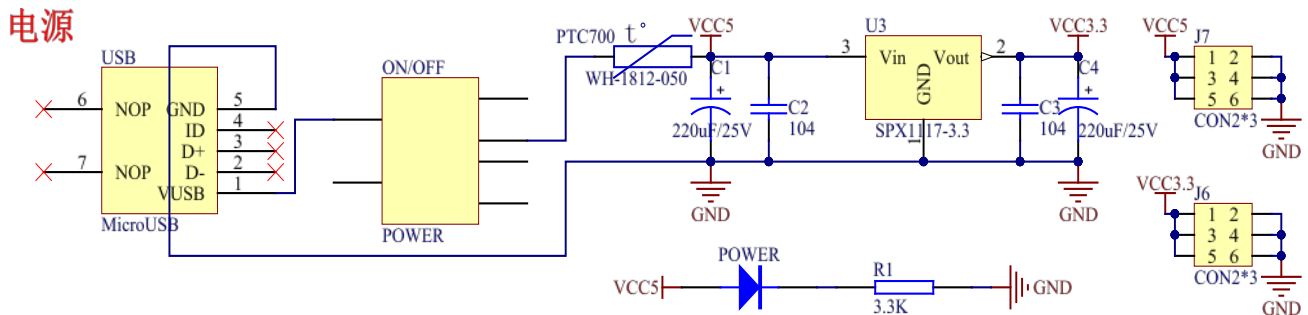


SDK1213 核心板使用 SC95F8523 作为主控芯片。同时外接 32KHz 低频晶振，可作为 BTM 低频时钟定时器振荡器。



3.2 电源模块

外接 MicroUSB 时，可通过自锁开关控制学习评估板电源，由电源指示灯 POWER 亮灭指示电源通断。通过 SPX1117-3.3 芯片将 V_{DD} 降压为 3.3V，并提供 V_{DD} 与 3.3V 的电源供给接口。



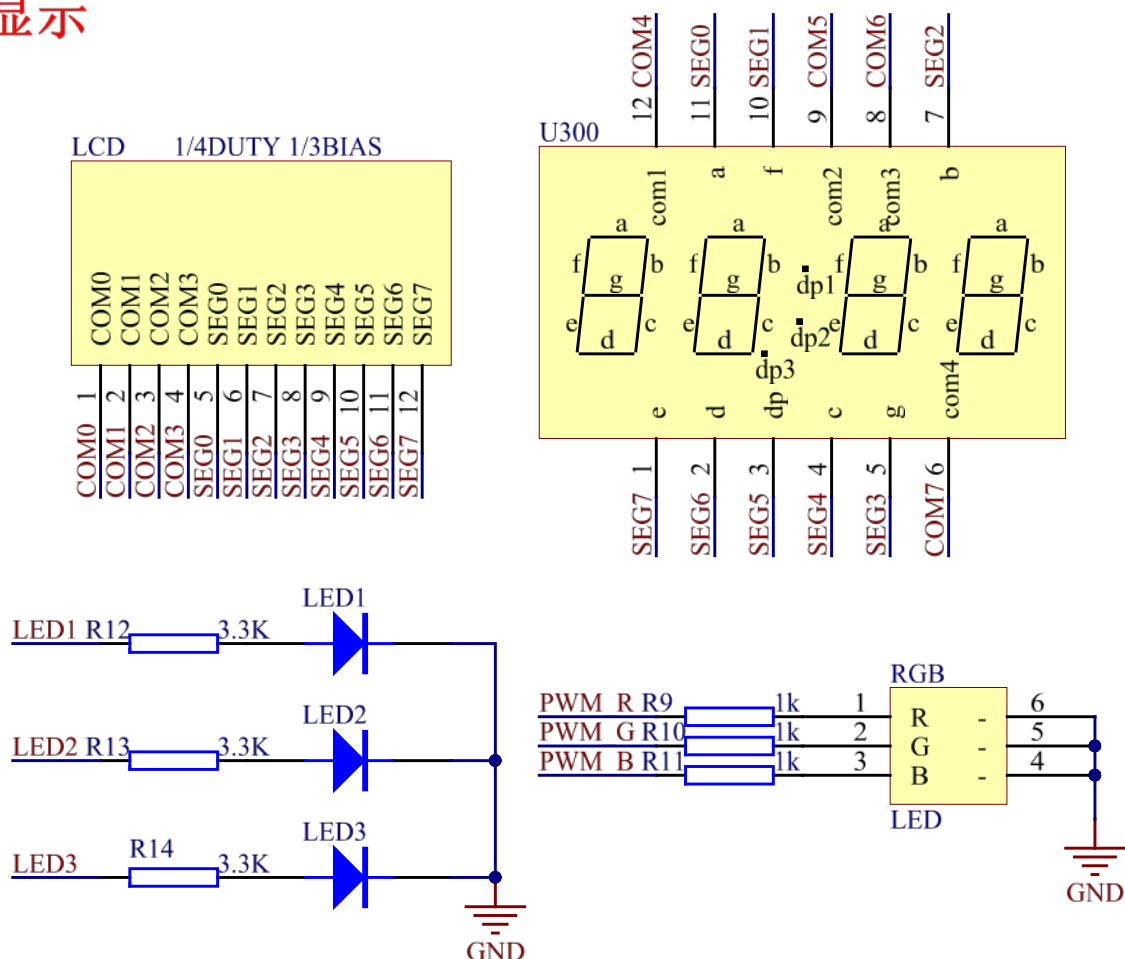
3.3 LED/RGB/LCD/数码管显示单元

SDK1010 提供三个 LED 灯，一个三色灯 RGB，一个 1/4duty、1/3bias LCD 显示屏与四位 LED 数码管。其中，LCD 显示屏与四位 LED 数码管共用 SEG 口，同一时刻只可使用 LCD 或 LED。

SDK1211/1212 核心板可控制三个 LED 灯，PWM 接口与三色灯 RGB 相连，可调节 PWM 占空比决定当前 RGB 颜色。SDK1211/1212 学习评估板上数码管和 LCD 屏的 SEG 及 COM 口与 MCU 硬件显示驱动电路相连。

SDK1213 学习评估板只能软件驱动数码管。

显示



3.4 按键与蜂鸣器

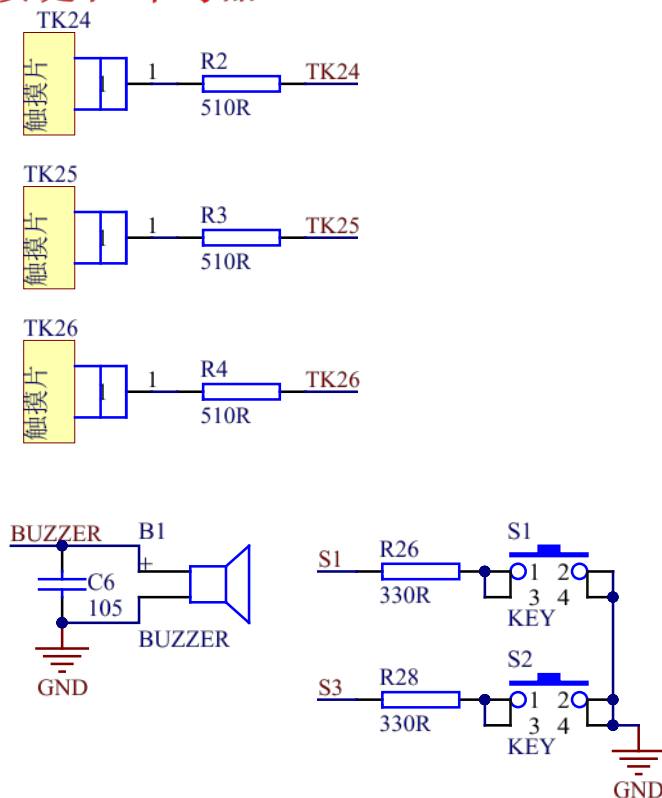
学习评估板提供三个 TouchKey 按键，帮助用户熟悉 MCU 的 TouchKey 功能。提供两个开关按键，按键 S1 与 Timer1 外部输入脚 T1 相连，按键 S2 与 Timer2 外部输入脚 T2 相连。各核心板使用 IO 口直接驱动无源蜂鸣器，但提供的 IO 不同。

SDK1211 核心板 MCU SC95F8617 提供三个 TouchKey 按键接口。

SDK1212 核心板 MCU SC95F8616 提供三个 TouchKey 按键接口。

SDK1213 核心板 MCU SC95F8523 提供一个 TouchKey 按键接口。

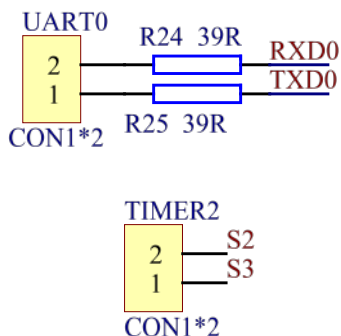
按键和蜂鸣器



3.5 UART0 与 TIMER2

MCU 支持一路全双工的串行口 UART0，通讯模式可选模式 0、模式 1、模式 3，TXD0 与 RXD0 引脚接出如下。其同时，将 Timer2 外部输入脚 T2 与外部捕获输入脚 T2EX 引出。

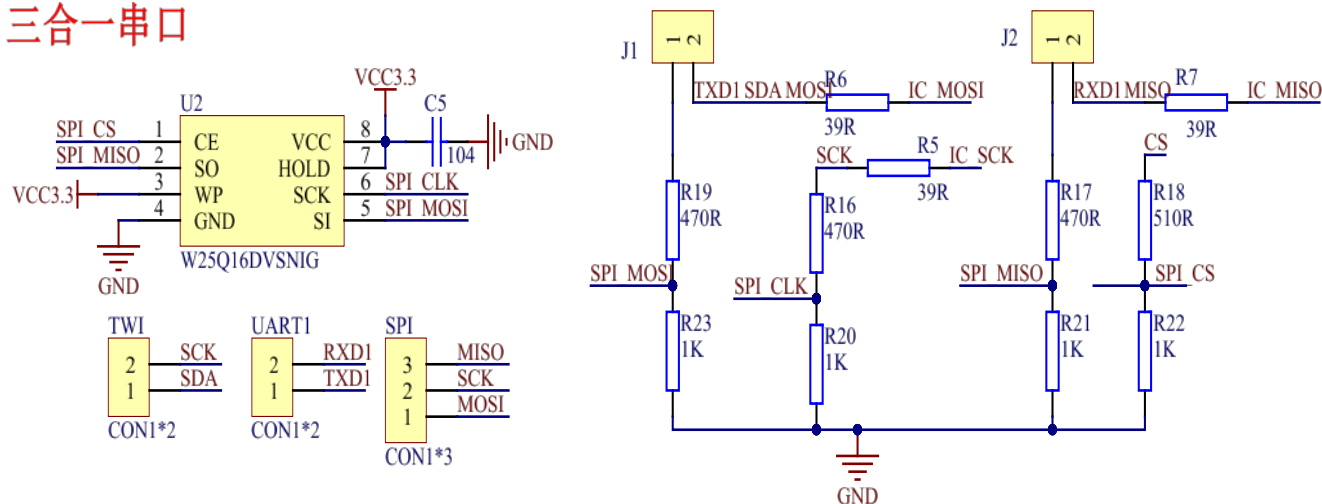
UART0与Timer2



3.6 三选一串行接口 USCI

MCU 内部集成三选一串行接口 USCI，可将 USCI 配置为 SPI、TWI 和 UART1 其中一种。TWI 兼容 IIC，但只可做从机。串口 UART1 支持模式 1 与模式 3。SPI 接口与 W25Q16 存储芯片相连，使用前需接上跳线帽。

三合一串口

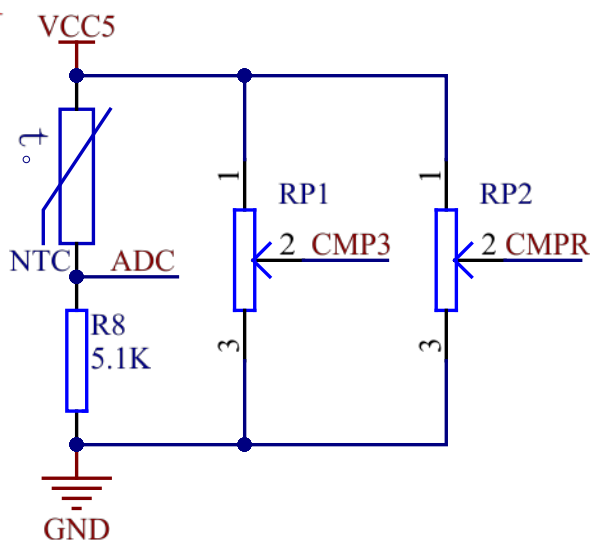


3.7 ADC 与 CMP

VCC 连接 NTC 热敏电阻与 5.1K 电阻 R8 到地，ADC 引脚采集 NTC 与 R8 之间电压，热敏电阻阻值随温度变化，ADC 采样值亦随之变化。贴片电阻 RP1 连接模拟比较器正输入端 CMP3，RP2 连接模拟比较器负输入端 CMPR，可调节 RP1、RP2 阻值使正负端电压变化。

SDK1213 不提供模拟比较器资源接口。

ADC/CMP



4.SDK121X 学习评估板示例

4.1 GPIO 示例

GPIO 端口可配置为三种模式，包括强推挽输出模式，带上拉的输入模式，高阻输入模式。GPIO 口模式配置由寄存器 PxCON 与 PxPH 控制，PxCON 控制 IO 为输入或输出模式，当端口作为输入时，每个 IO 端口带有由 PxPH 控制的内部上拉电阻。

强推挽输出模式下，能够提供持续的大电流驱动。带上拉的输入模式下，输入口上恒定接一个上拉电阻，仅当输入口上电平被拉低时，才会检测到低电平信号。

4.1.1 SDK1211/1212GPIO 示例

SDK1211 核心板 MCU SC95F8617 提供了最多 46 个可控制的双向 GPIO 端口，SDK1212 核心板 MCU SC95F8616 提供了最多 42 个可控制的双向 GPIO 端口。

GPIO 示例通过配置 GPIO 为强推挽输出模式，驱动 LED 灯，现象为 LED1 灯以固定的频率闪烁。

```
void SC_GPIO_Init(void)
{
    GPIO_Init(GPIO0, GPIO_PIN_6, GPIO_MODE_OUT_PP);
    /*GPIO_Init write here*/
}

void main(void)
{
    uint16_t i, j;
    SC_Init();

    while(1)
    {
        for(i=0; i<100; i++)                //延时一段时间
        {
            for(j=0; j<3000; j++);
        }

        GPIO_WriteLow(GPIO0, GPIO_PIN_6);    //P06写低
        for(i=0; i<100; i++)                //延时一段时间
        {
            for(j=0; j<3000; j++);
        }
    }
}
```

示例首先将与 LED 相连的 1 个 IO 端口配置为强推挽输出模式，在循环中将 IO 口高低电平状态以一定的延时不断改变。

4.1.2 SDK1213 GPIO 示例

SDK1213 核心板 MCU SC95F8523 提供了最多 26 个可控制的双向 GPIO 端口。

GPIO 示例通过在易码魔盒中单击管脚配置 GPIO 为强推挽输出模式，驱动 LED 灯，现象为 LED1 灯以固定的频率闪烁。



```

- void main(void)
3 {
  /*<UserCodeStart>*/
  uint16_t i,j;
  /*<UserCodeEnd>*/
  /*** MCU初始化函数 ***/
  SC_Init();
  /*<UserCodeStart>*////<SinOne-Tag><3>
  while(1)
3  {
    GPIO_WriteHigh(GPIO2,GPIO_PIN_6); //P26写高
    for(i=0;i<100;i++) //延时一段时间
3  {
      for(j=0;j<3000;j++);
    }

    GPIO_WriteLow(GPIO2,GPIO_PIN_6); //P26写低
    for(i=0;i<100;i++) //延时一段时间
3  {
      for(j=0;j<3000;j++);
    }
  }
  ///<UserCodeEnd>*////<SinOne-Tag><3>
- }
- }

```

示例首先将与 LED1 相连的 P2.6 端口配置为强推挽输出模式，在主循环中将 IO 口高低电平状态以一定的延时不断改变使 LED1 灯闪烁。

4.2 外部中断示例

外部中断 INT0~2，当外部中断口有中断条件发生时，外部中断就被触发。用户可以根据需要设成上升沿、下降沿或者双沿触发，可通过设置 SFR（INTxF 和 INTxR）来实现。用户可通过 IP 寄存器来设置每个中断的优先级级别。外部中断 INT0~2 还可以唤醒单片机的 STOP。

4.2.1 SDK1211/1212 INT 示例

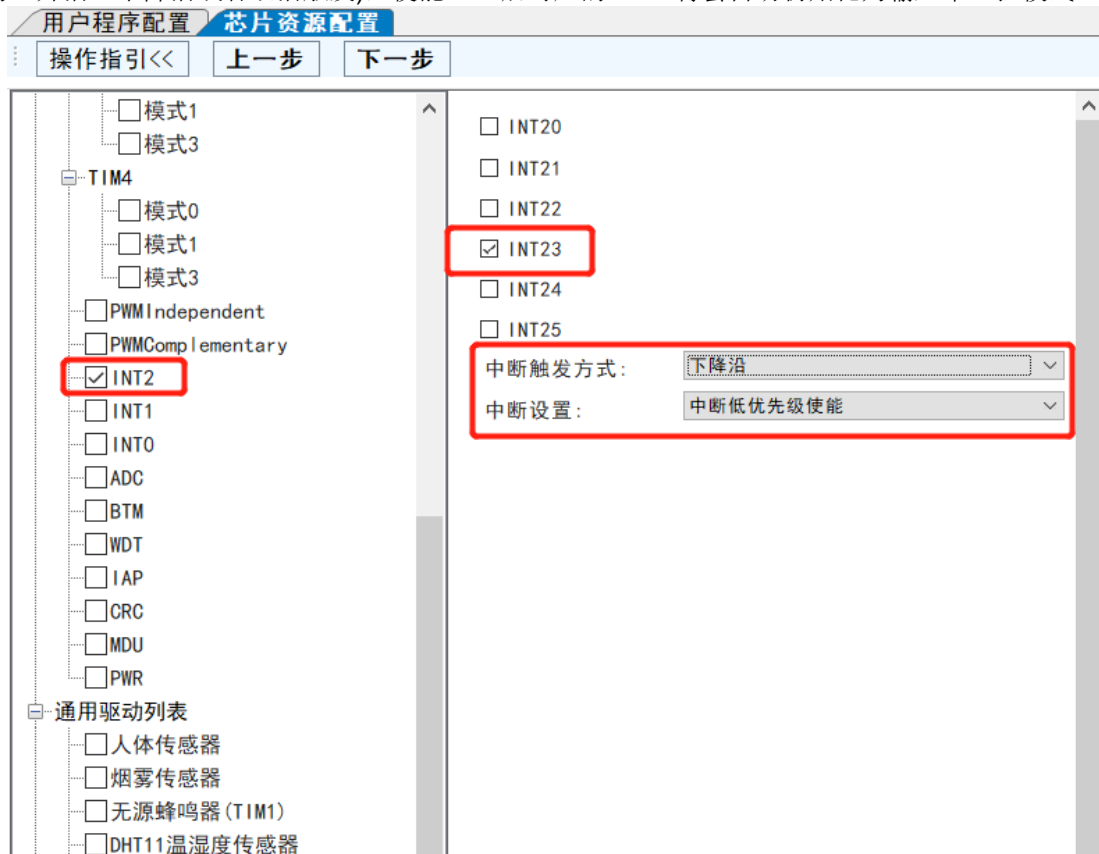
```
void SC_GPIO_Init(void)
{
    GPIO_Init(GPIO0, GPIO_PIN_6, GPIO_MODE_OUT_PP);
    /*GPIO_Init write here*/
}

void SC_INT_Init(void)
{
    GPIO_Init(GPIO0, GPIO_PIN_5, GPIO_MODE_IN_PU);
    INT0_SetTriggerMode(0x00|INT05, INT_TRIGGER_FALL_ONLY);
    INT0_ITConfig(ENABLE, LOW);
    /*INT_Init write here*/
}
```

外部中断示例通过将 IO 口 P05 配置为带上拉的输入模式，外部中断模式为下降沿触发的外部中断并使能外部中断 0 及总中断，而 P05 通过按键 S2 与地相连。当开关按键 S2 按下时产生一个下降沿，外部中断发生，此时在外部中断子程序中改变 LED1 灯的亮灭状态。主意，按键按下时产生的抖动可能会使灯的状态多次改变。

4.2.2 SDK1213 INT 示例

外部中断 INT0~2，当外部中断口有中断条件发生时，外部中断就被触发，INT0~2 中的各个 INT 共用中断和中断触发方式。用户在易码魔盒的芯片资源列表可以根据需要选择要使能的 INT 和设置中断优先级和中断触发方式(可设为上升沿、下降沿或者双沿触发)，使能 INT 后对应的 IO 口将会自动初始化为输入带上拉模式。





```

void INT2Interrupt()           interrupt 10
{
    /*<UserCodeStart>*/
    if (GPIO_ReadPin(GPIO2,GPIO_PIN_6))           //翻转IO
    {
        GPIO_WriteLow(GPIO2,GPIO_PIN_6);
    }
    else
    {
        GPIO_WriteHigh(GPIO2,GPIO_PIN_6);
    }
    /*<UserCodeEnd>*/
    /*INT2_it write here*/
}

```

外部中断示例使能 INT23 并将 P07 口配置为带上拉的输入模式，外部中断模式为下降沿触发的外部中断并使能外部中断，而 P2.6 设置为强推挽输出模式用于控制 LED1。当开关按键 S2 按下时产生一个下降沿，外部中断发生，此时在外部中断子程序中改变 LED1 灯的亮灭状态。注意，按键按下时产生的抖动可能会使灯的状态多次改变。

4.3 PWM 示例

4.3.1 SDK1211 /1212 PWM 示例

SC95F8617、SC95F8616 提供了最多 8 路共用周期、单独可调占空比的 12 位 PWM 输出。

PWM 具有的功能为：

1. 12 位 PWM 精度
2. 8 路 PWM 周期相同，但占空比可单独设置
3. 输出可设置正反向

PWM 可支持周期及占空比的调整，寄存器 PWMCG、PWMCON 控制 PWM 的状态及周期，各路 PWM 的打开及输出波形占空比可单独调整

示例通过 3 路输出的 PWM 波形控制 RGB 三色灯，现象为三色灯循环变色。

```
void SC_PWM_Init(void)
{
    PWM_Init(PWM_PRESSEL_FHRC_D1, 1199);
    PWM_OutputStateConfig(0x00|PWM40|PWM41|PWM53, PWM_OUTPUTSTATE_ENABLE);
    /*PWM波形无反相*/PWM_PolarityConfig(0x00|PWM40|PWM41|PWM42|PWM43|PWM50|PWM51|PWM52|PWM53, PWM_POLARITY_NON_INVERT);
    /*PWM40 独立模式*/PWM_IndependentModeConfig(PWM40, 0);
    /*PWM41 独立模式*/PWM_IndependentModeConfig(PWM41, 0);
    /*PWM53 独立模式*/PWM_IndependentModeConfig(PWM53, 0);
    PWM_ITConfig(DISABLE, LOW);
    PWM_Aligned_Mode_Select(PWM_Edge_Aligned_Mode);
    PWM_FaultDetectionConfig(DISABLE);
    PWM_FaultDetectionModeConfig(PWM_Latch_Mode, PWM_FaultDetectionVoltage_Low, PWM_WaveFilteringTime_Ous);
    PWM_Cmd(ENABLE);
    /*PWM_Init write here*/
}

void Delay_Some_Time(uint16_t Some_Time)
{
    uint16_t i, j;
    for(i=Some_Time; i>0; i--)
        for(j=100; j>0; j--);
}

void main(void)
{
    uint16_t i;
    SC_Init();

    while(1)
    {
        for(i=0; i<80; i++)
        {
            PWM_IndependentModeConfig(PWM40, i);
            Delay_Some_Time(30);
            PWM_IndependentModeConfig(PWM41, 80-i);
            Delay_Some_Time(30);
            PWM_IndependentModeConfig(PWM53, i);
            Delay_Some_Time(30);
        }

        for(i=0; i<80; i++)
        {
            PWM_IndependentModeConfig(PWM40, 80-i);
            Delay_Some_Time(30);
            PWM_IndependentModeConfig(PWM41, i);
            Delay_Some_Time(30);
            PWM_IndependentModeConfig(PWM53, 80-i);
            Delay_Some_Time(30);
        }
    }
}
```

示例首先设置 PWM 时钟源为 $F_{osc}/1$ ，周期为 $(1199+1)/(F_{osc}/1) = 37.5\mu s$ ，在主循环中，对与 RGB 相连的三路 PWM 使能，输出的 PWM 波形不反向，占空比随 for 循环不断改变，从而实现三色灯的循环变色。

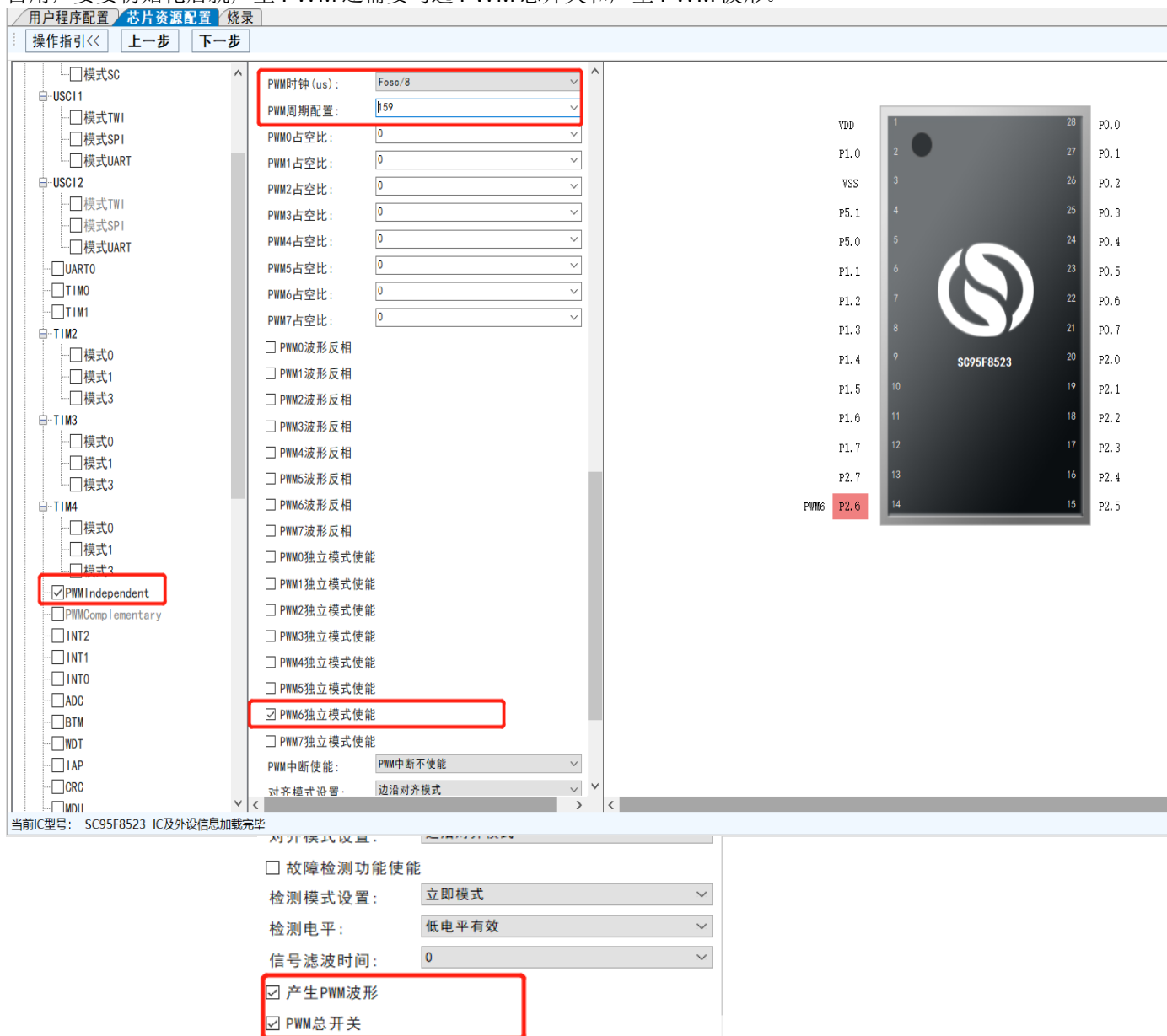
4.3.2 SDK1213 PWM 示例

SC95F8523 提供了最多 8 路共用周期、单独可调占空比的 16 位 PWM 输出。

PWM 具有的功能为：

1. 16 位 PWM 精度
2. 8 路 PWM 周期相同，但占空比可单独设置
3. 输出可设置正反向
4. 提供 1 个 PWM 溢出中断
5. 支持故障检测机制

用户在易码魔盒的芯片资源列表可以根据需要选择要使能的 PWM 和设置周期、占空比、中断优先级，PWM 周期配置值为 0~65535，且不小于 PWM 占空比。在配置菜单中使能了 PWM 那么对应得 IO 口红色表示 IO 已被占用，要要初始化后就产生 PWM 还需要勾选 PWM 总开关和产生 PWM 波形。



当前IC型号: SC95F8523 IC及外设信息加载完毕

对开模式设置:

- ☐ 故障检测功能使能
- 检测模式设置: 立即模式
- 检测电平: 低电平有效
- 信号滤波时间: 0
- ☒ 产生PWM波形
- ☒ PWM总开关


```

[  ****
void main(void)
{
  /*<UserCodeStart>*/
  uint8_t i = 0, Brightness = 0;
  /*<UserCodeEnd>*/
  /*** MCU初始化函数 ***/
  SC_Init();
  /*<UserCodeStart>*/  
//<SinOne-Tag><5>
  while(1)
  {
    for(i=0;i<47;i++)
    {
      Brightness = Brightness + BrightAdjust[i];
      PWM_IndependentModeConfig(PWM6, Brightness);
      Delay_Some_Time(100);
    }
    Delay_Some_Time(300);
    for(i=45;i>0;i--)
    {
      Brightness = Brightness - BrightAdjust[i];
      PWM_IndependentModeConfig(PWM6, Brightness);
      Delay_Some_Time(200);
    }
    PWM_IndependentModeConfig(PWM6, 0);
    Brightness = 0;
    Delay_Some_Time(3000);
  }
  /*<UserCodeEnd>*/  
//<SinOne-Tag><5>
}
]
```

示例首先设置 PWM 时钟源为 $F_{osc}/8$ ，周期为 $(159+1)/(F_{osc}/8) = 4\mu s$ ，在主循环中，不断得增减 PWM 的占空比使 LED1 呈现出呼吸灯效果。

4.4 TIMER0 示例

Timer0 是一个 16 位的定时计数器，具有计数方式和定时方式两种工作模式，特殊功能寄存器 TMOD 中有一个控制位 C/T0 来选择 T0 是定时器还是计数器，T0 本质上都是一个加法计数器，只是计数的来源不同。定时器的来源为系统时钟或者其分频时钟，但计数器的来源为外部管脚的输入脉冲。只有在 TR0=1 的时候，T0 才会被打开计数。

T0 有 4 种工作模式

1. 模式 0：13 位定时器/计数器模式
2. 模式 1：16 位定时器/计数器模式
3. 模式 2：8 位自动重载模式
4. 模式 3：两个 8 位定时器/计数器模式

4.4.1 SDK1211 /1212 TIMER0 示例

下面以 SDK1211 核心板为例，示例为演示 Timer0 的定时功能，现象为 LED1 灯每 600 毫秒亮灭状态改变一次。

```
void SC_TIM0_Init(void)
{
    TIMO_TimeBaseInit(TIM0_PRESSEL_FSYS_D12, TIMO_MODE_TIMER);
    TIMO_WorkModeConfig(TIM0_WORK_MODE0, 7999, 0);
    TIMO_ITConfig(ENABLE, LOW);
    TIMO_Cmd(ENABLE);
    /*TIM0_Init write here*/
}

void SC_GPIO_Init(void)
{
    GPIO_Init(GPIO0, GPIO_PIN_6, GPIO_MODE_OUT_PP);
    /*GPIO_Init write here*/
}

void main(void)
{
    SC_Init();
    while(1);
}

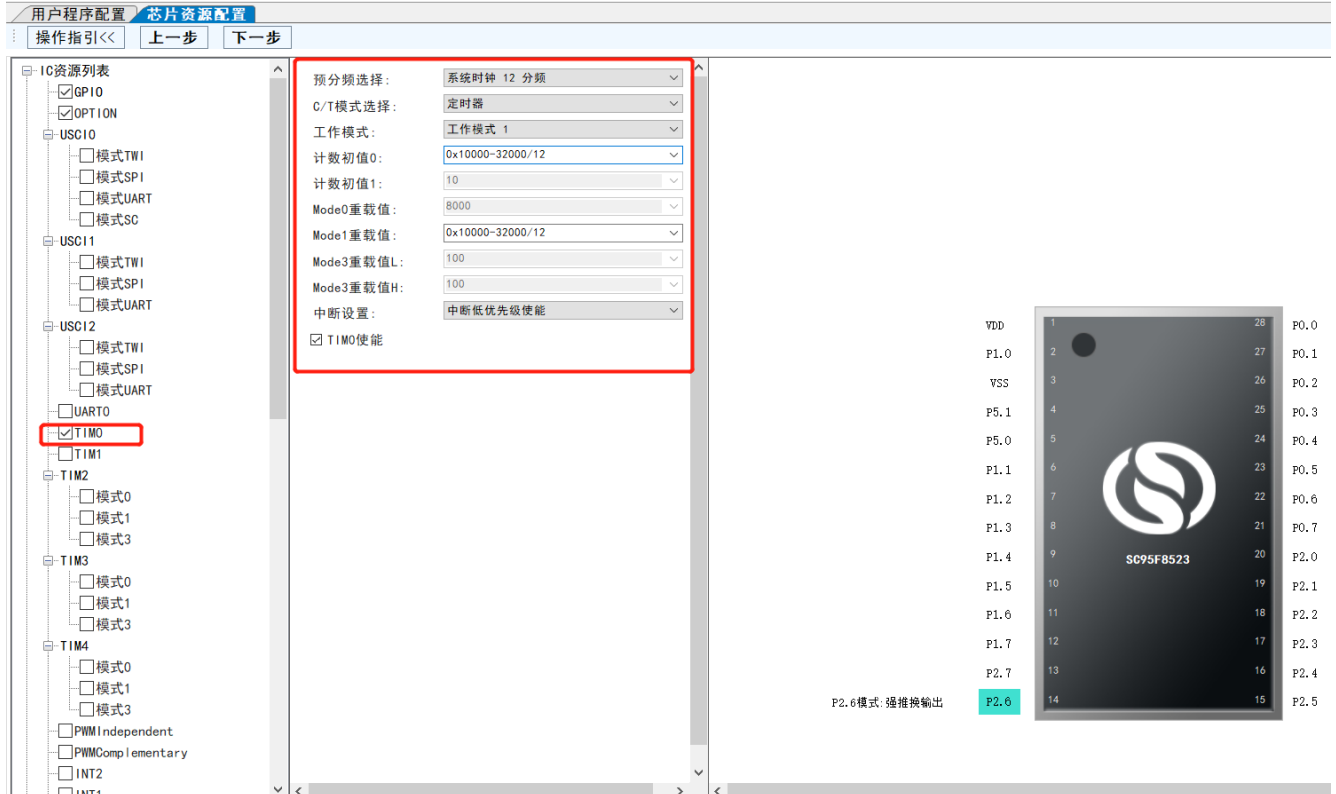
void Timer0Interrupt()           interrupt 1
{
    if(++TOCNT == 300)
    {
        TOCNT = 0;
        TIMO_Mode0SetReloadCounter(7999);
        if(GPIO_ReadPin(GPIO0, GPIO_PIN_6))
        {
            GPIO_WriteLow(GPIO0, GPIO_PIN_6);
        }
        else
        {
            GPIO_WriteHigh(GPIO0, GPIO_PIN_6);
        }
    }
}
```

T0 设置为定时器模式，工作方式模式 0，时钟源 12 分频，初值为（7999），即 3 毫秒进入一次中断，中断标志位累加到 200，即 600 毫秒时翻转 LED1 灯的状态。

4.4.2 SDK1213 TIMER0 示例

用户在易码魔盒的芯片资源列表可以根据需要选择要使用 TIM0，在 TIM0 的配置中要注意计数器初值以及重装值得范围，Mode0: 0~8191, Mode1: 0~65535, Mode2\3: 0~255。

示例为演示 Timer0 的定时功能，现象为 LED1 灯每 600 毫秒亮灭状态改变一次。



```
void Timer0Interrupt()           interrupt 1
{
    /*<UserCodeStart>*/
    static int i = 0;
    i ++;
    if(i >= 600)
    {
        i = 0;
        if(GPIO_ReadPin(GPIO2,GPIO_PIN_6))
        {
            GPIO_WriteLow(GPIO2, GPIO_PIN_6);
        }
        else
        {
            GPIO_WriteHigh(GPIO2, GPIO_PIN_6);
        }
    }
    /*<UserCodeEnd>*/
    TIM0_ModelSetReloadCounter(0x10000-32000/12);
    /*TIM0_it write here*/
}
```

示例首先设置 TIM0 时钟源为 $F_{osc}/12$ ，模式为定时器模式、工作模式 1，初值和重装值为 $0x10000-32000/12$ (定时 1ms)，并打开中断和使能 TIM0。在 TIM0 中断函数中，每隔 600ms 翻转 P26 控制 LED1 闪烁。

4.5 TIMER1 示例

Timer1 是一个 16 位的定时计数器，具有计数方式和定时方式两种工作模式，特殊功能寄存器 TMOD 中有一个控制位 C/T1 来选择 T1 是定时器还是计数器，T1 本质上都是一个加法计数器，只是计数的来源不同。定时器的来源为系统时钟或者其分频时钟，但计数器的来源为外部管脚的输入脉冲。只有在 TR1=1 的时候，T1 才会被打开计数。

T1 有 3 种工作模式

1. 模式 0：13 位定时器/计数器模式
2. 模式 1：16 位定时器/计数器模式
3. 模式 2：8 位自动重载模式

4.5.1 SDK1211 /1212 TIMER1 示例

下面以 SDK1211 核心板为例，示例为演示 Timer1 的计数功能，现象为每按下开关 S1，LED1 灯亮灭状态改变一次。

```
void SC_TIM1_Init(void)
{
    TIM1_TimeBaseInit(TIM1_PRESSEL_FSYS_D1, TIM1_MODE_COUNTER);
    TIM1_WorkModeConfig(TIM1_WORK_MODE1, 65535);
    TIM1_ITConfig(ENABLE, LOW);
    TIM1_Cmd(ENABLE);
    /*TIM1_Init write here*/
}

void SC_GPIO_Init(void)
{
    GPIO_Init(GPIO0, GPIO_PIN_6, GPIO_MODE_OUT_PP);
    GPIO_Init(GPIO0, GPIO_PIN_3, GPIO_MODE_IN_PU);
    /*GPIO_Init write here*/
}

void main(void)
{
    SC_Init();
    while(1)
    {
        if(T1FLG)
        {
            T1FLG = 0;
            if(GPIO_ReadPin(GPIO0, GPIO_PIN_6))
            {
                GPIO_WriteLow(GPIO0, GPIO_PIN_6);
            }
            else
            {
                GPIO_WriteHigh(GPIO0, GPIO_PIN_6);
            }
        }
    }
}

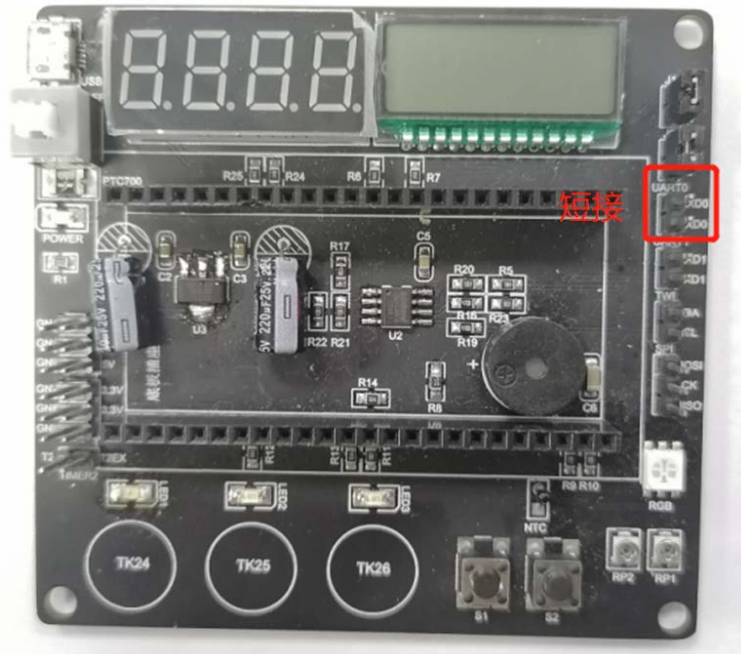
void Timer1Interrupt()    interrupt 3
{
    TIM1_Model1SetReloadCounter(65535);
    T1FLG = 1;
}
```

T1 设置为计数器模式，工作方式模式 1，时钟源不分频，初值为（65536-1），即计数一次进入中断，中断标志置起后，翻转 LED1 灯的状态。

4.5.2 SDK1213TIMER1 示例

用户在易码魔盒的芯片资源列表可以根据需要选择要使用 TIM1, TIM0 各设置项和 TIM1 一致。

示例为演示 Timer1 的计数功能，Timer0 控制 P12 翻转输出 1HZ 的方波，（短接 UART0 的 RX 和 TX）数码管显示计数值。



用户程序配置 芯片资源配置 烧录

操作指引<< 上一步 下一步

- USC12
 - ☐ 模式TWI
 - ☐ 模式SPI
 - ☐ 模式UART
- UART0
 - ☐ 模式0
 - ☐ 模式1
 - ☐ 模式3
- ☒ TIM0
- ☒ TIM1
- TIM2
 - ☐ 模式0
 - ☐ 模式1
 - ☐ 模式3
- TIM3
 - ☐ 模式0
 - ☐ 模式1
 - ☐ 模式3
- TIM4
 - ☐ 模式0
 - ☐ 模式1
 - ☐ 模式3
- ☐ PWMIndependent
- ☐ PWMComplementary
- ☐ INT2
- ☐ INT1
- ☐ INTO
- ☐ ADC
- ☐ BTM
- ☐ WDT
- ☐ IAP
- ☐ CRC
- ☐ MDU
- ☐ PWR
- 通用驱动列表
 - ☐ 人体传感器
 - ☐ 40 管脚 55

预分频选择: 系统时钟 12 分频

C/T模式选择: 计数器

工作模式: 工作模式 2

计数初值: 0

Mode0计数值重载: 0


Mode1计数值重载: 10000

中断设置: 中断低优先级使能

☒ TIM1 使能

数码管-数码管

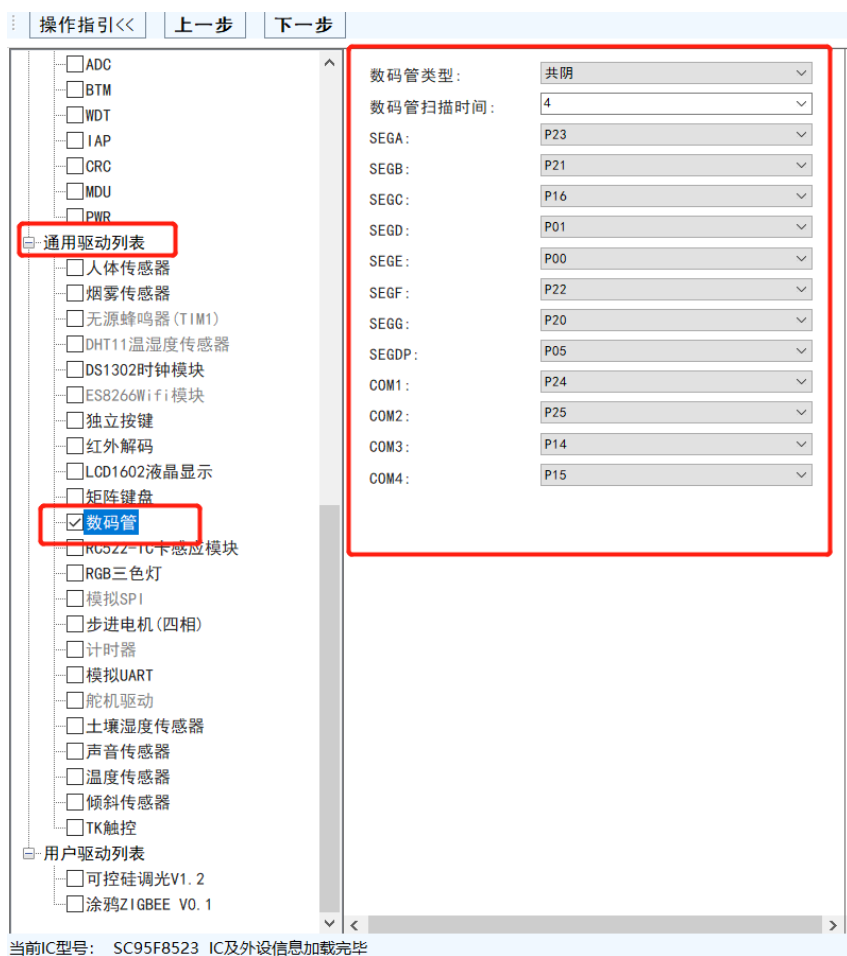
P23	SEG A	COM4	P15
P21	SEG B	COM3	P14
P16	SEG C	COM2	P25
P01	SEG D	COM1	P24
P00	SEG E	SEGDP	P05
P22	SEG F	SEG4	P20



P1.2模式: 强推挽输出

T1

VDD	1	28	P0.0
P1.0	2	27	P0.1
VSS	3	26	P0.2
P5.1	4	25	P0.3
P5.0	5	24	P0.4
P1.1	6	23	P0.5
P1.2	7	22	P0.6
P1.3	8	21	P0.7
P1.4	9	20	P2.0
P1.5	10	19	P2.1
P1.6	11	18	P2.2
P1.7	12	17	P2.3
P2.7	13	16	P2.4
P2.6	14	15	P2.5



当前IC型号: SC95F8523 IC及外设信息加载完毕

示例中的 Timer0 配置与 Timer0 例子中的一致，Timer1 设置为计数器且为工作模式 2，Timer1 被设为计数器后对应 T1 脚即 P1.3 被占用，P1.2 设置为强推挽输出模式。示例中会用到数码管，而在易码魔盒中自带数码管驱动，在通用驱动列表中勾选数码管并配置参数即可使用，数码管的驱动文件会自动加入工程中，调用接口函数即可。

```

void main(void)
{
    /*<UserCodeStart>*/
    /*<UserCodeEnd>*/
    /*** MCU初始化函数 ***/
    SC_Init();
    /*<UserCodeStart>*/  
/*<SinOne-Tag><13>  
SCD_NT_Display( 0 , 0 , 0 , 0 );  
TH1 = TL1 = 0;  
/*<UserCodeEnd>*/  
/*<SinOne-Tag><13>  
/*<UserCodeStart>*/  
/*<SinOne-Tag><11>  
while(1)  
{  
    /*<UserCodeStart>*/  
    SCD_NT_Display(  
        ((TH1<<8) + TL1)/1000%10,  
        ((TH1<<8) + TL1)/100%10,  
        ((TH1<<8) + TL1)/10%10,  
        ((TH1<<8) + TL1)%10  
    );  
    SCD_NT_Scan();  
    /*<UserCodeEnd>*/  
/*<SinOne-Tag><12>  
}  
/*<UserCodeEnd>*/  
/*<SinOne-Tag><11>  
}
    
```



```
void Timer0Interrupt()      interrupt 1
{
    /*<UserCodeStart>*/
    static int i = 0;
    i ++;
    if(i >= 500)
    {
        if(GPIO_ReadPin(GPIO1,GPIO_PIN_2))
        {
            GPIO_WriteLow(GPIO1,GPIO_PIN_2);
        }
        else
        {
            GPIO_WriteHigh(GPIO1,GPIO_PIN_2);
        }

        i = 0;
    }
    /*<UserCodeEnd>*/
    TIM0_ModelSetReloadCounter(0x10000-32000/12);
    /*TIM0_it write here*/
}
```

在示例程序中，Timer0 中不断翻转 P12 提供一个 1HZ 的方波，主循环不断将计数值刷新到数码管（SCD 开头的函数都源自通用用户驱动文件中）。

4.6 TIMER2 示例

Timer2 具有计数方式和定时方式两种工作模式。特殊功能寄存器 TXCON 中有一个控制位 C/T2 来选择 T2 是定时器还是计数器。它们本质上都是一个加法计数器，只是计数的来源不同。定时器的来源为系统时钟或者其分频时钟，但计数器的来源为外部管脚的输入脉冲。TRX 是 T2 在定时器/计数器模式计数的开关控制，只有在 TRX=1 的时候，T2 才会被打开计数。

计数器模式下，T2 管脚上的每一个脉冲，T2 的计数值分别增加 1。

定时器模式下，可通过特殊功能寄存器 TXMCON 来选择 T2 的计数来源是 fsys/12 或 fsys。

定时器/计数器 T2 有 4 种工作模式：

1. 模式 0：16 位捕获模式
2. 模式 1：16 位自动重载定时器模式
3. 模式 2：波特率发生器模式
4. 模式 3：可编程时钟输出模式

4.6.1 SDK1211 /1212 TIMER2 示例

示例将 Timer2 配置为 16 位捕获模式，四位数码管作显示用，用 Timer0 与 SCK 口产生的一个周期合适的方波，接在 T2EX 上，数码管显示该方波的周期，单位为 us。

```
void SC_TIM0_Init(void)
{
    TIM0_TimeBaseInit(TIM0_PRESSEL_FSYS_D1, TIM0_MODE_TIMER);
    TIM0_WorkModeConfig(TIM0_WORK_MODE2, 96, 0);
    TIM0_ITConfig(ENABLE, LOW);
    TIM0_Cmd(ENABLE);
    /*TIM0_Init write here*/
}

void SC_TIM2_Init(void)
{
    TIM2_TimeBaseInit(TIM2_MODE_TIMER, TIM2_COUNTDIRECTION_UP);
    TIM2_PrescalerSelection(TIM2_PRESSEL_FSYS_D1);
    TIM2_WorkMode0Config(0);
    TIM2_SetEXEN2(ENABLE);
    TIM2_ITConfig(ENABLE, LOW);
    TIM2_Cmd(ENABLE);
    /*TIM2_Init write here*/
}

void SC_DDIC_Init(void)
{
    DDIC_DMOD_Selcet(DMOD_LED);
    DDIC_Init(DDIC_DUTYCYCLE_D4, 0x00, 0x00|DDIC_PIN_X4|DDIC_PIN_X5|DDIC_PIN_X6|DDIC_PIN_X7, 0x00|DDIC_PIN_X2|DDIC_PIN_X3|DDIC_PIN_X4|DDIC_PIN_X5, 0xf0);
    DDIC_OutputPinOfDutycycleD4(SEG0_27COM4_7);
    DDIC_Cmd(ENABLE);
    /*DDIC_Init write here*/
}

void main(void)
{
    SC_Init();
    while(1)
    {
        if(T0Flag120ms)
        {
            T0Flag120ms = 0;
            if(CaptureState & 0X80)
            {
                LedDataTab[0] = (uint16_t)Cycle / 1000;    //显示数据装载到数组
                LedDataTab[1] = (uint16_t)Cycle / 100 % 10;
                LedDataTab[2] = (uint16_t)Cycle / 10 % 10;
                LedDataTab[3] = (uint16_t)Cycle % 10;
                CaptureState=0;
            }
            Led_Display();
        }
    }
}
```

```

void Timer2Interrupt()      interrupt 5
{
    static uint16_t CaptureValue1, CaptureValue2;
    static uint16_t ReloadCount = 0;
    if((CaptureState & 0X80) == 0)
    {
        if(TIM2_GetFlagStatus(TIM2_FLAG_TF2))
        {
            if(CaptureState & 0X40)
                ReloadCount++;          //ReloadCount为T2计数溢出次数
        }
        if(TIM2_GetFlagStatus(TIM2_FLAG_EXF2))
        {
            if(CaptureState & 0X40)
            {
                CaptureValue2 = ((uint16_t)RCAPXH << 8) + RCAPXL; //CaptureValue2为捕获第二个下降沿时记录的捕获值
                Cycle = ((65536*ReloadCount) + CaptureValue2 - CaptureValue1) / 16; //捕获的方波周期为 (T2溢出时间+第二次捕获值-第一次捕获值)
                TLX = 0X00; //清除计数值
                THX = 0X00;
                CaptureState |= 0X80;
                ReloadCount = 0;
            }
            else
            {
                CaptureValue1 = ((uint16_t)RCAPXH << 8) + RCAPXL; //CaptureValue1为捕获第一个下降沿时记录的捕获值
                CaptureState = 0X40;
                ReloadCount = 0;
            }
        }
    }
    if(TIM2_GetFlagStatus(TIM2_FLAG_TF2))
    {
        TIM2_ClearFlag(TIM2_FLAG_TF2);
    }
    if(TIM2_GetFlagStatus(TIM2_FLAG_EXF2))
    {
        TIM2_ClearFlag(TIM2_FLAG_EXF2);
    }
}

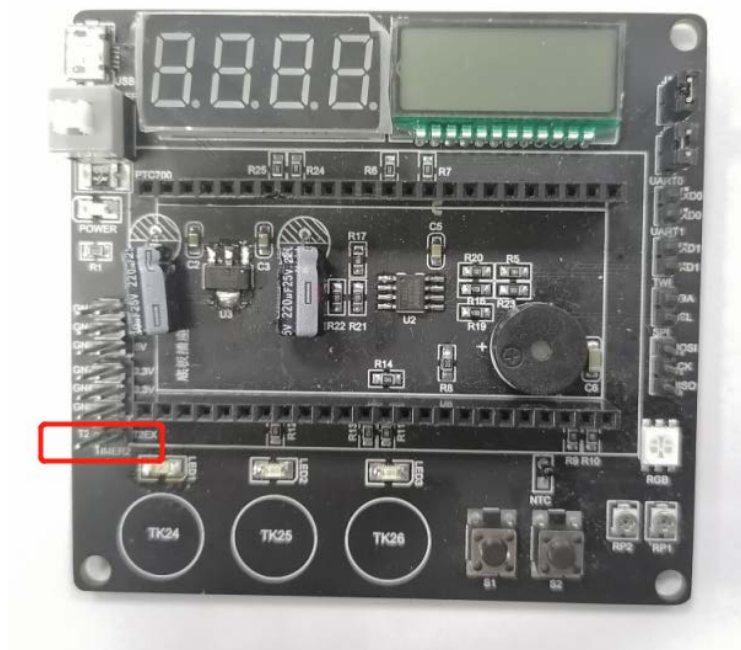
```

T2EX 上的下降沿触发 T2 捕获模式的中断，在中断服务程序中记录当前计数值，在下一个下降沿到来后记录计数值，此时计数值相减即可求周期。

4.6.2 SDK1213 TIMER2 示例

SC95F8523/7523 有 5 个 TIMER,其中 TIMER2\3\4 都支持输入捕获模式。

示例将 Timer2 设为捕获模式下降沿有效，Timer0 控制 P0.7 输出周期 20ms 的方波，将 Timer2 接口短接，数码管上显示该方波的周期，单位毫秒。





该示例用到 TIM0、TIM1、TIM2 三个定时器在配置上无太大差异，TIM2 设置捕获口下降沿有效后 T2EX 被占用用作输入捕获，Timer1 控制 P0.7 输出周期 20ms 的方波而 TIM0 用于计时。（数码管外设驱动在通用驱动列表中）

```

void main(void)
{
    /*<UserCodeStart>*/
    /*<UserCodeEnd>*/
    /*** MCU初始化函数 ***/
    SC_Init();
    /*<UserCodeStart>*/  
/*<SinOne-Tag><17>  
time0_lms = (void)0;  
/*<UserCodeEnd>*/  
/*<SinOne-Tag><17>  
/*<UserCodeStart>*/  
/*<SinOne-Tag><18>  
SCD_NT_Display( 0 , 0 , 0 , 0 );  
/*<UserCodeEnd>*/  
/*<SinOne-Tag><18>  
/*<UserCodeStart>*/  
/*<SinOne-Tag><14>  
while(1)  
{  
  
    /*<UserCodeStart>*/  
    /*<SinOne-Tag><16>  
    SCD_NT_Scan();  
    /*<UserCodeEnd>*/  
    /*<SinOne-Tag><16>  
}  
/*<UserCodeEnd>*/  
/*<SinOne-Tag><14>  
}
    
```

```
64
65 void Timer1Interrupt()      interrupt 3
66 {
67     /*<UserCodeStart>*/
68     if(GPIO_ReadPin(GPIO0,GPIO_PIN_7))
69     {
70         GPIO_WriteLow(GPIO0,GPIO_PIN_7);
71     }
72     else
73     {
74         GPIO_WriteHigh(GPIO0,GPIO_PIN_7);
75     }
76     /*<UserCodeEnd>*/
77     TIM1_ModelSetReloadCounter(0x10000-320000/12);
78     /*TIM1_it write here*/
79 }
80
```

在主循环中只进行数码管刷新，而在 Timer1 中断函数中翻转 P0.7 输出方波。

```
void Timer2Interrupt()      interrupt 5
{
    /*<UserCodeStart>*/
    static int flagEXF = 0;
    if(TIM2_GetFlagStatus(TIM2_FLAG_EXF2))
    {
        TIM2_ClearFlag(TIM2_FLAG_EXF2);
        if(flagEXF == 0)
        {
            time0_lms = 0;
            flagEXF = 1;
        }
        else
        {
            time0_lms = time0_lms%10 >= 5 ? time0_lms + 10:time0_lms;
            SCD_NT_Display(
                time0_lms/10000%10,
                time0_lms/1000%10,
                time0_lms/100%10,
                time0_lms/10%10
            );
            flagEXF = 0;
        }
    }
    if(TIM2_GetFlagStatus(TIM2_FLAG_TF2))
    {
        TIM2_ClearFlag(TIM2_FLAG_TF2);
    }
    /*<UserCodeEnd>*/
    /*TIM2_it write here*/
}

```

在 Timer2 中断函数中等待下降沿到来获取两次下降沿的间隔时间并将时间刷新到数码管上。（注意：TIM2 中断标志位需要手动清除）

4.7 LED 显示驱动示例

SC95F8617、SC95F8616 内部集成了硬件的 LCD/LED 显示驱动电路，可方便用户实现 LCD 和 LED 的显示驱动。其主要特点如下：

1. LCD 和 LED 显示驱动二选一；
2. LCD 和 LED 显示驱动共用相关 IO 口和寄存器。

LED 显示驱动功能如下：

1. 4 种显示驱动模式可选：8 X 24、6 X 26、5 X 27、或 4X 28 段；
2. seg 口驱动能力 4 级可选；
3. 显示驱动电路可选择内建 32K LRC 或外部 32K 振荡器作为时钟源，帧频约为 64Hz。

示例 code 启动 LED 硬件驱动电路，现象为 LED 数码管每秒的显示值加 1，从 0~F 循环显示。

```
void SC_DDIC_Init(void)
{
    DDIC_DMOD_Selcet(DMOD_LED);
    DDIC_Init(DDIC_DUTYCYCLE_D4, 0x00, 0x00|DDIC_PIN_X4|DDIC_PIN_X5|DDIC_PIN_X6|DDIC_PIN_X7, 0x00|DDIC_PIN_X2|DDIC_PIN_X3|DDIC_PIN_X4|DDIC_PIN_X5, 0xf0);
    DDIC_OutputPinOfDutycycleD4(SEG0_27COM4_7);
    DDIC_Cmd( ENABLE);
    /*DDIC_Init write here*/
}

void main(void)
{
    uint8_t LedDisplayNum = 0;
    SC_Init();
    while(1)
    {
        if(TOFlag10ms)
        {
            TOFlag10ms = 0;
            Led_Display();
        }
        if(TOFlag1s)
        {
            TOFlag1s = 0;
            LedDisplayNum++; //显示数据加一
            if(LedDisplayNum==16)
            {
                LedDisplayNum = 0;
            }
            LedDataTab[0] = LedDisplayNum; //显示数据装载到数组
            LedDataTab[1] = LedDisplayNum;
            LedDataTab[2] = LedDisplayNum;
            LedDataTab[3] = LedDisplayNum;
        }
    }
}

void Timer0Interrupt() interrupt 1
{
    TIM0_Model1SetReloadCounter(32000);
    TOFlag10msCount++;
    TOFlag1sCount++;
    if(TOFlag10msCount >= 10)
    {
        TOFlag10msCount = 0;
        TOFlag10ms = 1;
    }
    if(TOFlag1sCount >= 1000)
    {
        TOFlag1sCount = 0;
        TOFlag1s = 1;
    }
    /*TIM0_it write here*/
}
```

在主函数中，配置 T0 为定时器模式，计数值为 32000，定时时间为 10ms，置起标志 TOFlag10msCount，累加中断标志中断标志 TOFlag10msCount，TOFlag1sCount 累加 1000 次，即 1 秒使显示值加 1。开启 LED 硬件扫描，在主循环中，每隔 10 毫秒更新 LED 数码管显示数据。SDK1010 主板 LED 与 LCD 共用 SEG 口，使用 LED 时，LCD 会显示乱码。

4.8 LCD 显示驱动示例

SC95F8617、SC95F8616 内部集成了硬件的 LCD/LED 显示驱动电路。

LCD 显示驱动功能如下：

1. 4 种显示驱动模式可选：8 X 24、6 X 26、5 X 27、或 4X 28 段；
2. 2 种偏置方式可选：1/4 Bias 和 1/3 Bias；
3. com 口驱动能力 4 级可选；
4. 显示驱动电路可选择内建 32K LRC 或外部 32K 振荡器作为时钟源，帧频约为 64Hz。

示例 code 启动 LCD 硬件驱动电路，现象为 LCD 屏每秒的显示值加 1，从 0~F 循环显示。

```
void SC_DDIC_Init(void)
{
    DDIC_DMOD_Selcet(DMOD_LCD);
    DDIC_Init(DDIC_DUTYCYCLE_D4, 0x00, 0x00|DDIC_PIN_X4|DDIC_PIN_X5|DDIC_PIN_X6|DDIC_PIN_X7, 0x00|DDIC_PIN_X2|DDIC_PIN_X3|DDIC_PIN_X4|DDIC_PIN_X5, 0x0f);
    DDIC_LCDConfig(4, DDIC_ResSel_100K, DDIC_BIAS_D3);
    DDIC_OutputPinOfDutycycleD4(SEG4_27COM0_3);
    DDIC_Cmd( ENABLE);
    /*DDIC_Init write here*/
}

void main(void)
{
    uint8_t LcdDisplayNum = 0;
    SC_Init();

    while(1)
    {
        if(TOFlag10ms)
        {
            TOFlag10ms = 0;
            Lcd_Display();
        }
        if(TOFlag1s)
        {
            TOFlag1s = 0;
            LcdDisplayNum++; //显示数据加一
            if(LcdDisplayNum==16)
            {
                LcdDisplayNum = 0;
            }
            LcdDataTab[0] = LcdDisplayNum; //显示数据装载到数组
            LcdDataTab[1] = LcdDisplayNum;
            LcdDataTab[2] = LcdDisplayNum;
            LcdDataTab[3] = LcdDisplayNum;
        }
    }
}

void Timer0Interrupt() interrupt 1
{
    TIMO_ModelSetReloadCounter(32000);
    TOFlag10msCount++;
    TOFlag1sCount++;
    if(TOFlag10msCount >= 10)
    {
        TOFlag10msCount = 0;
        TOFlag10ms = 1;
    }
    if(TOFlag1sCount >= 1000)
    {
        TOFlag1sCount = 0;
        TOFlag1s = 1;
    }
    /*TIMO_it write here*/
}
```

在主函数中，配置 T0 为定时器模式，计数值为 32000，定时时间为 10ms，置起标志 TOFlag10msCount，累加中断标志中断标志 TOFlag10msCount，TOFlag1sCount 累加 1000 次，即 1 秒使显示值加 1。开启 LCD 硬件扫描，在主循环中，每隔 10 毫秒更新 LCD 屏显示数据。

4.9 模拟比较器示例

4.9.1 SDK1211 /1212 模拟比较器示例

SC95F8617、SC95F8616 内建一个模拟比较器，此比较器具有四个模拟信号正输入端：CMP0~3，可通过 CMPIS [1:0]切换选择。负输入端电压可通过 CMPRF[3:0]切换为 CMPR 脚上的外部电压或内部的 16 档比较电压中的一种。

通过 CMPIM[1:0]可以方便的设定比较器的中断模式，当 CMPIM[1:0]所设定的中断条件发生时比较器中断标志 CMPIF 会被置 1，该中断标志需要软件清除。

示例 code 启动模拟比较器功能，通过调节 RP1 电阻可控制比较器正端输入电压（P43 口电压），调节 RP2 电阻可控制比较器负端比较电压（P44 口电压），现象为当正端电压大于负端电压时，LED1 灯亮，反之 LED2 灯亮。

```
void SC_ACMP_Init(void)
{
    ACMP_Init(ACMP_VREF_EXTERNAL, ACMP_CHANNEL_3);
    ACMP_SetTriggerMode(ACMP_TRIGGER_RISE_ONLY);
    ACMP_ITConfig(ENABLE, LOW);
    ACMP_Cmd(ENABLE);
    /*ACMP_Init write here*/
}

void SC_GPIO_Init(void)
{
    GPIO_Init(GPIO0, GPIO_PIN_6, GPIO_MODE_OUT_PP);
    GPIO_Init(GPIO5, GPIO_PIN_2, GPIO_MODE_OUT_PP);
    /*GPIO_Init write here*/
}

void main(void)
{
    SC_Init();
    while(1)
    {
        if(ACMP_GetFlagStatus(ACMP_FLAG_CMPSTA))    //读比较器输出状态
        {
            GPIO_WriteHigh(GPIO0, GPIO_PIN_6);
            GPIO_WriteLow(GPIO5, GPIO_PIN_2);
        }
        else
        {
            GPIO_WriteHigh(GPIO5, GPIO_PIN_2);
            GPIO_WriteLow(GPIO0, GPIO_PIN_6);
        }
    }
}

void ACMPInterrupt()    interrupt 12
{
    ACMP_ClearFlag();
    /*ACMP_it write here*/
}
```

示例将 LED1 灯与 LED2 灯配置为强推挽模式，开启模拟比较器功能，正端输入电压端口选择 CMP3，负端比较电压选择外部 CMPR 电压，模拟比较器中断触发条件为双沿中断。通过读取输出状态位的状态，控制 LED1 灯、LED2 灯的亮灭。

4.10 低频时钟定时器 BTM 示例

SC95F8617、SC95F8616、SC95F8523 内建一个频率为 32kHz 的 RC 及 32.768kHz 晶体振荡电路，都可作为低频时钟定时器 Base Timer 的时钟源。该振荡器直接连接一个 Base Timer，可以把 CPU 从 STOP mode 唤醒，并且产生中断。

4.10.1 SDK1211/1212 BTM 示例

示例 code 开启 BTM，现象为 LED1 灯每 0.25 秒亮灭状态改变一次。

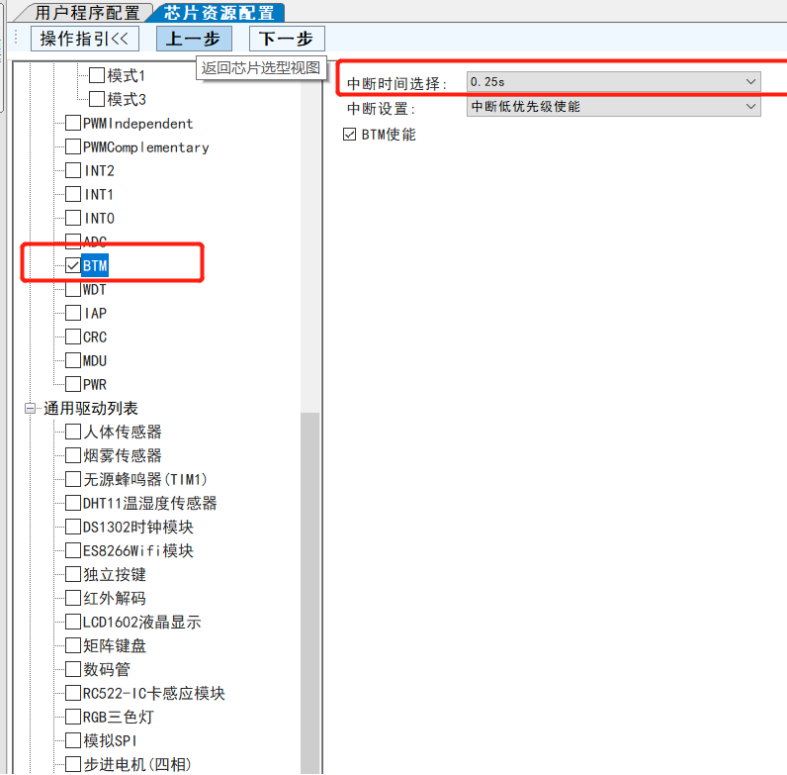
```
void main(void)
{
    SC_Init();
    while(1)
    {
        if(GPIO_ReadPin(GPIO5,GPIO_PIN_0))           //P50为外部晶振输入脚，外部晶振使能时为低
        {
            GPIO_WriteHigh(GPIO5,GPIO_PIN_2);
        }
        else
        {
            GPIO_WriteLow(GPIO5,GPIO_PIN_2);
        }
    }
}


void BTMInterrupt()           interrupt 9
{
    if(GPIO_ReadPin(GPIO0,GPIO_PIN_6))
    {
        GPIO_WriteLow(GPIO0,GPIO_PIN_6);
    }
    else
    {
        GPIO_WriteHigh(GPIO0,GPIO_PIN_6);
    }
    /*BTM_it write here*/
}
```

示例 code 在 option 项中勾选使用外部 32.768 晶体振荡电路，此时内建 32k Hz RC 停止工作。BTM 设置为每 0.25 秒产生一个中断，中断标志置起后翻转 LED1 灯状态。若 option 项不勾选外部晶振，此时内部低频 RC 工作，LED1 灯依旧 0.25 秒改变一次状态，由于外部低频晶振不工作，OSCI (P50) 为 IO 口，且为上拉状态，故 LED2 灯将常亮。

4.10.2 SDK1213 BTM 示例

示例 code 开启 BTM，现象为 LED1 灯每 0.25 秒亮灭状态改变一次。





```

4 void BTMInterrupt()           interrupt 9
5 {
6     /*<UserCodeStart>*/
7     if (GPIO_ReadPin(GPIO2,GPIO_PIN_6))
8     {
9         GPIO_WriteLow(GPIO2,GPIO_PIN_6);
10    }
11    else
12    {
13        GPIO_WriteHigh(GPIO2,GPIO_PIN_6);
14    }
15    /*<UserCodeEnd>*/
16    /*BTM_it write here*/
17 }
  
```

示例中设置 BTM 中断时间为 0.25s，则每隔 0.25s 进入一次 BTM 中断，在中断中翻转 P26 使 LED1 闪烁，若烧录界面 option 项不勾选外部晶振，此时内部低频 RC 工作，BTM 时钟由于内部低频 RC 提供。

4.11 ADC 示例

4.11.1 SDK1211/1212 ADC 示例

SC95F8617、SC95F8616 内建一个 12-bit 17 通道的高精度逐次逼近型 ADC，外部的 ADC 通道和 IO 口的其它功能复用。内部的一路可接至 $1/4 V_{DD}$ ，配合内部 2.4V 参考电压用于测量 V_{DD} 电压。

ADC 的参考电压可以有 2 种选择：

1. V_{DD} 管脚(即直接是内部的 V_{DD})；
2. 内部 Regulator 输出的参考电压精准的 2.048V 或 1.024V。

注意： f_{ADC} 直接由内部 f_{HRC} 分频所得，用户在配置时要注意 ADC 的时钟频率 f_{ADC} 不可大于系统时钟的频率 f_{SYS} ，否则会引起 ADC 转换结果异常！

以 SC95F8617 为例，利用 ADC 采集电压，当热敏电阻 NTC 的阻值随温度改变时，ADC 所采集的电压亦随之改变，由热敏电阻的温度特性可通过当前的电压值得当前的温度值。现象为 LED 数码管显示当前室温，用手触摸 NTC，数码管显示温度亦随之变化。

```
void SC_TIMO_Init(void)
{
    TIMO_TimeBaseInit(TIMO_PRESSEL_FSYS_D1, TIMO_MODE_TIMER);
    TIMO_WorkModeConfig(TIMO_WORK_MODE1, 0, 0);
    TIMO_ITConfig(ENABLE, LOW);
    TIMO_Cmd(ENABLE);
    /*TIMO_Init write here*/
}

void SC_ADC_Init(void)
{
    ADC_Init(ADC_Cycle_6Cycle);
    /*AIN14口模式设置*/ADC_EAINConfig(ADC_EAIN_14, ENABLE);
    ADC_ChannelConfig(ADC_CHANNEL_14, ENABLE);
    ADC_ITConfig(DISABLE, LOW);
    ADC_Cmd(ENABLE);
    /*ADC_Init write here*/
}

void SC_DDIC_Init(void)
{
    DDIC_DMOD_Selcet(DMOD_LED);
    DDIC_Init(DDIC_DUTYCYCLE_D4, 0x00, 0x00|DDIC_PIN_X4|DDIC_PIN_X5|DDIC_PIN_X6|DDIC_PIN_X7, 0x00|DDIC_PIN_X2|DDIC_PIN_X3|DDIC_PIN_X4|DDIC_PIN_X5, 0xf0);
    DDIC_OutputPinOfDutycycleD4(SEG0_27COM4_7);
    DDIC_Cmd(ENABLE);
    /*DDIC_Init write here*/
}

void main(void)
{
    /*** MCU初始化函数 ***/
    SC_Init();
    while(1)
    {
        if(TOFlag500ms)
        {
            TOFlag500ms = 0;
            GetADCValue();           //得到温度值的ADC转换值
            GetTemperature();        //通过数组ADCValueToTemp[]求得温度值
            Led_Display();
        }
    }
}
```

示例利用 T0 的定时功能，定时更新 LED 数码管的数据显示，开启 ADC 功能，选择通道 14 为采样通道，设置采样时间为 6 个采样时钟周期。

4.11.2 SDK1213 ADC 示例


SC95F8523 内建一个 12-bit 11 通道的高精度逐次逼近型 ADC，外部的 ADC 通道和 IO 口的其它功能复用。内部的一路可接至 1/4 VDD，配合内部 2.048V 或者 1.024V 参考电压用于测量 VDD 电压。

ADC 的参考电压可以有 2 种选择：

1. VDD 管脚(即直接是内部的 VDD);
2. 内部 Regulator 输出的参考电压精准的 2.048V 或 1.024V。

注意：fADC 直接由内部 fHRC 分频所得，用户在配置时要注意 ADC 的时钟频率 fADC 不可大于系统时钟的频率 fSYS，否则会引起 ADC 转换结果异常！

该示例程序 ADC 采集 NTC 热敏电阻电压，转换为当前室温，数码管显示室温。



The screenshot shows the configuration interface for the SC95F8523 microcontroller. The 'ADC' module is selected in the 'IC Resource List'. The 'Sampling Time' is set to '3 ADC sampling clock periods'. The 'Sampling Port' is set to 'AIN6'. The 'Interrupt Setting' is 'Interrupt Disabled'. The 'ADC Function Switch Selection' is checked. The 'General Driver List' shows various sensors and modules. On the right, a diagram of the SC95F8523 chip shows the pin connections for the ADC and the 4-digit 7-segment display.

用户在界面设置完 ADC 配置参数后在函数中调用 ADC_GetConversionValue 即可获取当前采样口的 AD 值，示例中的为 AIN6。

```
uint16_t GetADCValue()
{
    uint16_t i;
    uint16_t ADC_ValueSum = 0;
    uint16_t ADC_Value;
    uint16_t value_max = 0, value_min = 5000;
    for(i=0; i<10; i++)
    {
        ADC_StartConversion();
        while(!ADC_GetFlagStatus());
        ADC_ClearFlag();
        ADC_Value = ADC_GetConversionValue();
        if(ADC_Value >= value_max) value_max = ADC_Value;
        if(ADC_Value <= value_min) value_min = ADC_Value;
        ADC_ValueSum = ADC_ValueSum + ADC_Value;
    }
    ADC_Value = (ADC_ValueSum - value_max - value_min) / 8;
    return ADC_Value;
}
```

GetADCValue 函数多次获取 ADC 值除去最大最小值求取平均值，用来保证 ADC 采样值的可靠。

```
void GetTemperature(uint16_t ADC_Value)
{
    uint16_t code ADCValueToTemp[100] =
    {
        248    ,    262    ,    277    ,    293    ,    310    ,    327    ,    345    ,
        425    ,    447    ,    470    ,    494    ,    518    ,    543    ,    570    ,
        683    ,    713    ,    744    ,    776    ,    809    ,    842    ,    877    ,
        1022   ,    1060   ,    1098   ,    1137   ,    1177   ,    1217   ,    1258   ,
        1425   ,    1468   ,    1511   ,    1554   ,    1598   ,    1641   ,    1685   ,
        1860   ,    1903   ,    1946   ,    1990   ,    2033   ,    2075   ,    2117   ,
        2283   ,    2324   ,    2364   ,    2403   ,    2442   ,    2481   ,    2519   ,
        2664   ,    2699   ,    2734   ,    2767   ,    2800   ,    2833   ,    2864   ,
        2985   ,    3013   ,    3041   ,    3069   ,    3095   ,    3121   ,    3147   ,
        3242   ,    3265   ,    3286   ,    3308   ,    3328   ,    3348   ,    3368   ,
    };

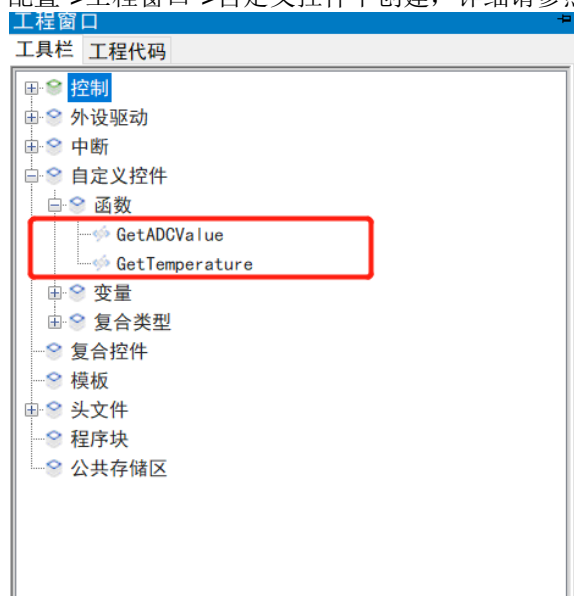
    int TEMP,j = 0;
    uint8_t ADC_High,ADC_Low;
    for(j=0;j<100;j++)
    {
        if((ADC_Value >= ADCValueToTemp[j]) && (ADC_Value < ADCValueToTemp[j+1]))
        {
            TEMP = j;
            break;
        }
    }
    if(TEMP < 15)
    {
        TEMP = 15 - TEMP;
    }
    else
    {
        TEMP = TEMP - 15;
    }

    ADC_High = TEMP / 10;
    ADC_Low  = TEMP % 10;

    SCD_NT_Display(16,ADC_High,ADC_Low,12);
}
```

GetTemperature 利用 ADC 值求取温度值并且将温度结果刷新到数码管上。

（函数可以在易码魔盒用户程序配置->工程窗口->自定义控件中创建，详细请参照易码魔盒使用说明）



4.12 串口 UART0 示例

MCU支持一个全双工的串行口UART0，UART0的功能及特性如下：

1. 三种通讯模式可选：模式 0、模式 1 和模式 3；
2. 可选择定时器 1 或定时器 2 作为波特率发生器；
3. 发送和接收完成可产生中断 RI/TI，该中断标志需要软件清除。

4.12.1 SDK1211/1212 UART0 示例

以 SDK1211 示例 code 为例，使用 UART0 与串口上位机通信，波特率为 9600，现象为上位机显示“UART0 is OK!”。

```
void SC_UART0_Init(void)
{
    GPIO_Init(GPIO2, GPIO_PIN_0,GPIO_MODE_IN_PU);
    GPIO_Init(GPIO2, GPIO_PIN_1,GPIO_MODE_IN_PU);
    /*模式1初始化*/UART0_Init(32000000, 9600, UART0_Mode_10B, UART0_CLOCK_TIMER1, UART0_RX_DISABLE);
    UART0_ITConfig(ENABLE, LOW);
    /*UART0_Init write here*/
}

.....

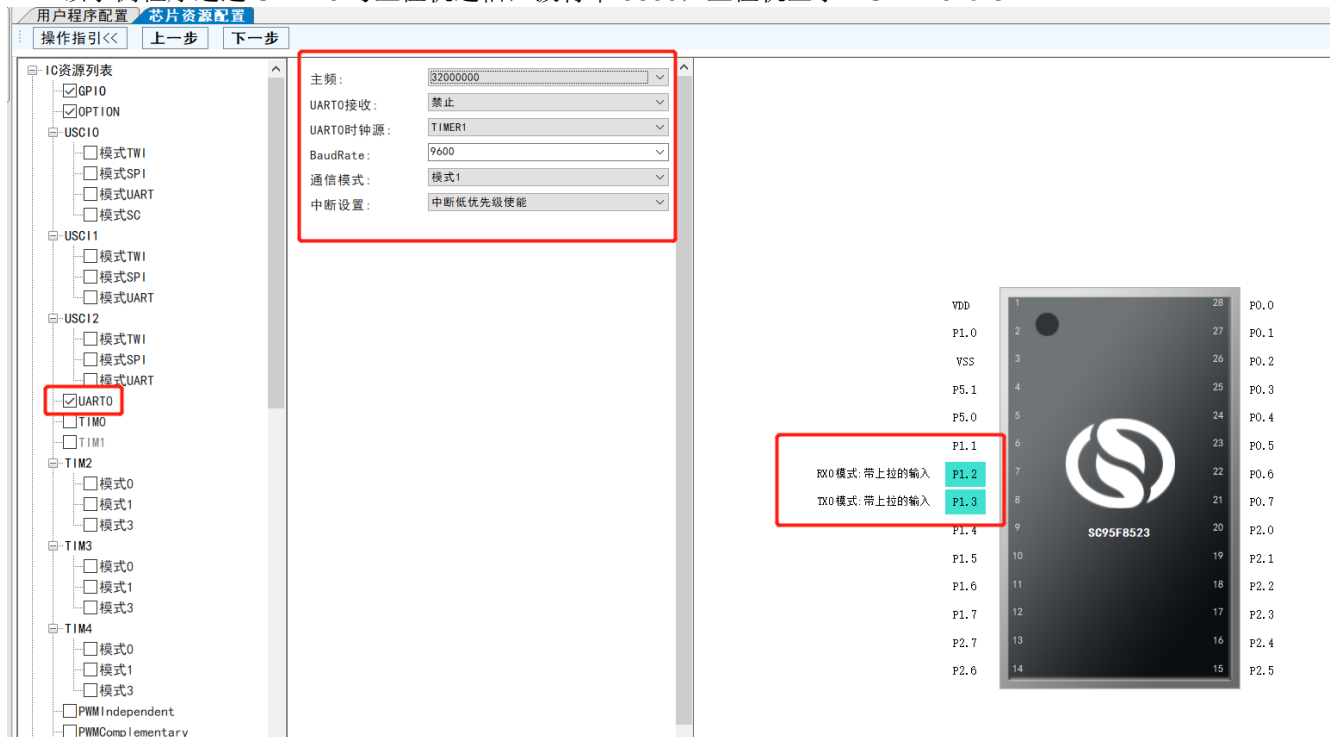
void main(void)
{
    SC_Init();
    while(1)
    {
        printf("UART0 is OK!\n");
    }
}

char putchar(char c)
{
    UART0_SendData8(c);
    while(!Uart0SendFlag);
    Uart0SendFlag = 0;
    return c;
}
```

示例配置 UART0 为模式 1，使能 UART0 接收，使用 T1 作为波特率发生器，波特率为 9600。在主循环中不断往上位机发送字符串“UART0 is OK!”。

4.12.2 SDK1213 UART0 示例

该示例程序通过 UART0 与上位机通信，波特率 9600，上位机显示“UART0 is OK”！



在该示例中，勾选使用 UART0（勾选 UART0 后对应的 RX0、TX0 会自动设置位输入上拉模式），设置主频 32M（主频用于计算波特率参数，若主频错误波特率也会出错），设置 BaudRate 为 9600，使用 TIMER1 作为波特率发生器，使用模式 1（10 位全双工异步通信）。

```

- void main(void)
- {
-     /*<UserCodeStart>*/
-     /*<UserCodeEnd>*/
-     /*** MCU初始化函数 ***/
-     SC_Init();
-     /*<UserCodeStart>*/  
/*SinOne-Tag><18>
-     printf("\r\n\r\n\r\n\r\n\r\nUART0 OK\r\n\r\n\r\n\r\n");
-     /*<UserCodeEnd>*/  
/*SinOne-Tag><18>
-     /*<UserCodeStart>*/  
/*SinOne-Tag><17>
-     while(1)
-     {
-     }
-     /*<UserCodeEnd>*/  
/*SinOne-Tag><17>
- }

- char putchar(char c)
- {
-     UART0_SendData8(c);
-     while(!Uart0SendFlag);
-     Uart0SendFlag = 0;
-     return c;
- }
- /*<UserCodeEnd>*/

```

在程序中初始化后即发送“UART0 OK”。

4.13 IAP 示例

IAP 操作空间范围有两种模式可选：

IC 整个 ROM 空间范围都可进行 IAP 操作，主要用作远程程序更新使用。

IAP 操作空间选择作为 Code Option 在编程器写入 IC 时选择。

示例 code 通过 IAP 将数组的值写进指定 ROM 区域，再读出指定 ROM 区域的值与数组的值比较，若相同，则 UART0 发送 ROM START 与 END 至上位机，若不相同，上位机将显示详细信息。

```
void SC_UART0_Init(void)
{
    GPIO_Init(GPIO2, GPIO_PIN_0, GPIO_MODE_IN_PU);
    GPIO_Init(GPIO2, GPIO_PIN_1, GPIO_MODE_IN_PU);
    /*模式1初始化*/UART0_Init(32000000, 9600, UART0_Mode_10B, UART0_CLOCK_TIMER1, UART0_RX_DISABLE);
    UART0_ITConfig(ENABLE, LOW);
    /*UART0_Init write here*/
}

int main(void)
{
    uint16_t i, j=4;
    uint16_t IAP_Value = 0;
    SC_Init();
    IAP_SectorErase(IAP_MEMTYPE_ROM, 64512, 0xf0); //擦除扇区
    printf("IAP ROM START-----\n");

    for(i=64512; i<64522; i++, j++)
    {
        IAP_ProgramByte(i, j, IAP_MEMTYPE_ROM, 0xf0);
        IAP_Value = IAP_ReadByte(i, IAP_MEMTYPE_ROM);
        if(IAP_Value!= j)
        {
            printf("IAP Fail! Error message is\n");
            printf("Address:%u\n", i);
            printf("Data:%d\n", IAP_Value);
        }
    }
    printf("IAP ROM END-----\n");
    while(1);
}
```

示例使用 IAP 功能在 ROM 地址上，进行读写操作。读出的值若与写的值不同，上位机将显示不同处的地址，以及对应地址内的数据。

4.14 乘除法器示例

4.14.1 SDK1211/1212 MDU 示例

SC95F8617/95F8616 提供了 1 个 16 位的乘除法器，由扩展累加器 EXA0~EXA3、扩展 B 寄存器 EXB 和运算控制寄存器 OPERCON 组成。可取代软件进行 16 位×16 位乘法运算和 32 位/16 位除法运算。

示例 code 开启乘除法器功能，现象为若乘法结果正确，则 LED1 灯亮，若除法结果正确，则 LED2 灯亮。

```
void SC_GPIO_Init(void)
{
    GPIO_Init(GPIO0, GPIO_PIN_6, GPIO_MODE_OUT_PP);
    GPIO_Init(GPIO5, GPIO_PIN_2, GPIO_MODE_OUT_PP);
    /*GPIO_Init write here*/
}

void SC_MDU_Init(void)
{
    MDU_MultiplicationConfig(10000, 50);    //设置乘数与被乘数10000*50
    MDU_StartOperation();                  //启动运算
    /*MDU_Init write here*/
}

void main(void)
{
    SC_Init();
    if(MDU_GetProduct() == 500000)
    {
        GPIO_WriteHigh(GPIO0, GPIO_PIN_6); //结果正确LED1灯亮
    }

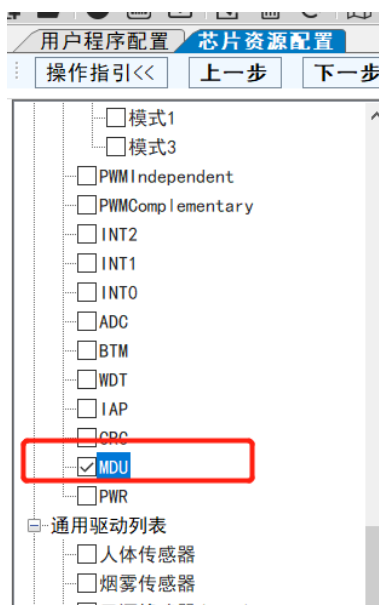
    MDU_DivisionConfig(10000, 50);         //设置被除数与除数10000/50
    MDU_StartOperation();                  //启动运算
    if((MDU_GetQuotient() == 200) && (MDU_GetRemainder() == 0))
    {
        GPIO_WriteHigh(GPIO5, GPIO_PIN_2); //结果正确LED2灯亮
    }

    while(1);
}
```

示例将 LED1 与 LED2 配置为强推挽输出模式。乘法功能中入参为乘数与被乘数，除法功能中入参为被除数与除数。由函数返回值可得乘除法结果。

4.14.2 SDK1213 MDU 示例

该示例主要乘除法器的使用，在程序中利用两个数相乘和相除，若乘除法结果都正确，则 LED1 闪烁两次



在易码魔盒中勾选 MDU 软件就会自动将 MDU 的 BSP 包加入工程，调用 MDU_DivisionConfig 装载计算值、MDU_StartOperation 开启计算，MDU_GetProduct 获取乘法结果、MDU_GetQuotient 获取商、MDU_GetRemainder 获取余数。

```
void main(void)
{
    /*<UserCodeStart>*/
    long i = 0;
    /*<UserCodeEnd>*/
    /** MCU初始化函数 **/
    SC_Init();
    /*<UserCodeStart>*/  
    i = 300000;  
    while(i--);  
    MDU_MultiplicationConfig(100, 50);    //设置乘数与被乘数10000*50  
    MDU_StartOperation();                //启动运算  
    if(MDU_GetProduct() == 5000)  
    {  
        GPIO_WriteHigh(GPIO2, GPIO_PIN_6);    //结果正确LED1灯亮  
    }  
    i = 300000;  
    while(i--);  
    GPIO_WriteLow(GPIO2, GPIO_PIN_6);  
    i = 300000;  
    while(i--);  
    MDU_DivisionConfig(10000, 50);        //设置被除数与除数10000/50  
    MDU_StartOperation();                //启动运算  
    if((MDU_GetQuotient()==200) && (MDU_GetRemainder()==0))  
    {  
        GPIO_WriteHigh(GPIO2, GPIO_PIN_6);    //结果正确LED1灯亮  
    }  
    i = 300000;  
    while(i--);  
    GPIO_WriteLow(GPIO2, GPIO_PIN_6);  
    /*<UserCodeEnd>*/  
    /*<UserCodeStart>*/  
    while(1)  
    {  
    }  
    /*<UserCodeEnd>*/  
}
```

在主循环开头先计算乘除法，若结果正确使 LED1 闪烁两次

4.15 USCI1_UART 示例

SDK1211/1212/1213 核心板 MCU 内部集成了三选一串行接口电路（简称 USCI），可配置为串口模式 UART。UART 模式可工作在模式 1（10 位全双工异步通信）和模式 3（11 位全双工异步通信）。

4.15.1 SDK1211/1212 UART1 示例

以 SDK1211 为例，示例 code 使用 UART1 与串口上位机通信，使能接收功能，波特率为 9600，当上位机发生数据到 MCU，MCU 将同样的数据发送到上位机。

```
void SC_USCI1_Init(void)
{
    GPIO_Init(GPIO1, GPIO_PIN_1, GPIO_MODE_IN_PU);
    GPIO_Init(GPIO1, GPIO_PIN_3, GPIO_MODE_IN_PU);
    USCI1_ITConfig(ENABLE, LOW);
    USCI1_UART_Init(32000000, 9600, USCI1_UART_Mode_10B, USCI1_UART_RX_ENABLE);
    /*USCI1_Init write here*/
}

void USCI1Interrupt()           interrupt 15
{
    if(USCI1_GetFlagStatus(USCI1_UART_FLAG_TI))
    {
        USCI1_ClearFlag(USCI1_UART_FLAG_TI);
    }
    if(USCI1_GetFlagStatus(USCI1_UART_FLAG_RI))
    {
        USCI1_ClearFlag(USCI1_UART_FLAG_RI);
        USCI1_UART_SendData8(USCI1_UART_ReceiveData8());
    }
}
```

4.15.2 SDK1213 USCI1_UART 示例

该示例使用 UART1 与上位机通信，波特率 9600，上位机发送数据到 MCU，MCU 将所接收到的内容发出。



在易码魔盒中勾选 USCI1-模式 UART，将 USCI1 设置为 UART 功能，UART 模式和 UART0 的设置项一致，可参考 UART0，这里 UART 设置为模式 1、波特率为 9600。

```
- void USCI1Interrupt()           interrupt 15
- {
-     /*<UserCodeStart>*/
-     if(USCI1_GetFlagStatus(USCI1_UART_FLAG_TI))
-     {
-         USCI1_ClearFlag(USCI1_UART_FLAG_TI);
-     }
-     if(USCI1_GetFlagStatus(USCI1_UART_FLAG_RI))
-     {
-         USCI1_ClearFlag(USCI1_UART_FLAG_RI);
-         USCI1_UART_SendData8(USCI1_UART_ReceiveData8());
-     }
-     /*<UserCodeEnd>*/
-     /*USCI1_it write here*/
- }
```

在 USCI1 中断函数中接收到了数据后又立即发出去。

4.16 SPI1 示例

SDK1211/1212/1213 核心板 MCU 内部集成了三选一串行接口电路（简称 USCI），可配置为高速串行通信接口 SPI，允许 MCU 与外围设备(包括其它 MCU)进行全双工，同步串行通信。

SPI 可配置为主模式或从属模式中的一种。以 SDK1211 为例，SPI 模块的配置和初始化通过设置 US0CON0 寄存器(SPI 控制寄存器)和 US0CON1(SPI 状态寄存器)来完成。配置完成后，通过设置 US0CON2,US0CON3,SSDAT(SPI 数据寄存器)来完成数据传送。通过软件设置 US0CON0 寄存器的 CPOL 位和 CPHA 位，用户可以选择 SPI 时钟极性和相位的四种组合方式。

4.16.1 SDK1211/1212 SPI1 示例

示例 code 通过 SPI 操作 W25Q16，将数组数据 0~9 写入 W25Q16，再将 W25Q16 中的值读出来并通过 UART0 发送给上位机，现象为上位机显示 0~9。

```
void SC_USCI1_Init(void)
{
    GPIO_Init(GPIO1, GPIO_PIN_1,GPIO_MODE_IN_PU);
    GPIO_Init(GPIO1, GPIO_PIN_3,GPIO_MODE_IN_PU);
    USCI1_SPI_Init(USCI1_SPI_FIRSTBIT_MSB,USCI1_SPI_BAUDRATEPRESCALER_128,USCI1_SPI_MODE_MASTER,USCI1_SPI_CLOCKPOLARITY_LOW,
    USCI1_SPI_CLOCKPHASE_1EDGE,USCI1_SPI_TXE_ENINT,USCI1_SPI_DATA8);
    USCI1_ITConfig(ENABLE,LOW);
    USCI1_SPI_Cmd(ENABLE);
    /*USCI1_Init write here*/
}

char putchar(char c)    //重定义该函数
{
    UART0_SendData8(c);
    while(!Uart0SendFlag);
    Uart0SendFlag = 0;
    return c;
}

void main(void)
{
    uint16_t i;
    SC_Init();
    for(i=0;i<10;i++)
    {
        Send_DATA[i] = i;    //发送数组赋值
        Rec_DATA[i] = 2;    //接收数组赋值
    }

    W25_SectorErase(0x000000);    //W25Q16擦除0x0000起始的4k空间
    SPI_Flash_Write_NoCheck(Send_DATA,0x000000,10);    //往0x0000地址写10个数，数值为Send_DATA[i]内值
    SPI_Flash_Read(Rec_DATA,0x000000,10);    //从0x0000读10个数，存放到Rec_DATA[10]

    while(1)
    {
        printf("\n Rec_DATA[10] value = \n{");    //串口打印Rec_DATA数组值，观察是否为写入值

        for(i=0;i<10;i++)
        {
            printf("%c",Rec_DATA[i]+0x30);
        }
        printf("}");
    }
}

void USCI1Interrupt()    interrupt 15
{
    if(USCI1_GetFlagStatus(USCI1_SPI_FLAG_SPIF))
    {
        USCI1_ClearFlag(USCI1_SPI_FLAG_SPIF);
        SPIFLG = 1;
    }

    if(USCI1_GetFlagStatus(USCI1_SPI_FLAG_TXE))
    {
        USCI1_ClearFlag(USCI1_SPI_FLAG_TXE);
    }
}
```

示例使用 UART0 与串口上位机通信，SPI 配置 SCK 在空闲状态下为低电平，第一沿采集数据，SPI 时钟速率为 fsys /128，MSB 优先发送。将数组 Send_DATA[10]的值写进 W25Q16，再从 W25Q16 将这些值读出来放在数组 Rec_DATA[10]中，通过 UART0 将 Rec_DATA[10]发出。

4.16.2 SDK1213 SPI1 示例

该示例和 SDK1211/1212 的示例功能一致。



在易码魔盒中勾选 USC11-模式 SPI 将 USC11 设置为 SPI 功能，根据 W25Q16 的 SPI 通讯规则进行设置，数据传输高位优先、SPI 作为主设备等等。（注意:为了与 W25Q16 通讯稳定 SPI 速度不宜太快）

```

*****
void main(void)
{
    /*<UserCodeStart>*/
    uint16_t i;
    for(i=0;i<10;i++)
    {
        Send_DATA[i] = i; //发送数组赋值
        Rec_DATA[i] = 0; //接收数组赋值
    }

    /*<UserCodeEnd>*/
    /** MCU初始化函数 **/
    SC_Init();
    /*<UserCodeStart>*/<SinOne-Tag><22>
    W25_SectorErase(0x000000); //W25Q16擦除0x0000起始的4k空间
    SPI_Flash_Write_NoCheck(Send_DATA,0x000000,10); //往0x0000地址写10个数，数值为Send_DATA[i]内值
    SPI_Flash_Read(Rec_DATA,0x000000,10); //从0x0000读10个数，存放到Rec_DATA[10]
    /*<UserCodeEnd>*/<SinOne-Tag><22>
    /*<UserCodeStart>*/<SinOne-Tag><21>
    while(1)
    {
        printf("\r\n\r\n\r\n\r\n Rec_DATA[10] value = \n{"); //串口打印Rec_DATA数组值，观察是否为写入值

        for(i=0;i<10;i++)
        {
            printf("%d", (int)Rec_DATA[i]); //转化为ASCII值
        }
        printf("\r\n\r\n\r\n\r\n");
    }
    /*<UserCodeEnd>*/<SinOne-Tag><21>
}
}

```

在程序中将数组 Send_DATA[10] 的值写进 W25Q16，再从 W25Q16 将这些值读出来放在数组 Rec_DATA[10] 中，通过 UART0 将 Rec_DATA[10] 发出。

4.17 触控按键电路 TOUCH KEY 示例

SC95F8617/95F8616/95F8523 内建一个 31(SC95F8523 为 23)通道的高灵敏度电容触控按键 Touch Key Sensor 电路, 可实现隔空按键触控、接近感应等操作。可配置为高灵敏度模式或高可靠模式, 高灵敏度模式可实现隔空按键触控、接近感应等灵敏度要求较高的触控应用。高可靠模式具有很强的抗干扰能力, 可通过 10V 动态 CS 测试。

用户通过使用赛元提供的高灵敏度触控按键库文件, 可快速简单实现隔空按键、接近感应等功能。

4.17.1 SDK1211/1212 TK 示例

示例 code 调用赛元高灵敏度触控库, 配置三个 TouchKey 按键, 按下触控按键, 按键上方对应的 LED 灯亮, 同时蜂鸣器鸣响。

```
/*入口参数: void
******/
void SC_Init(void)
{
    SC_GPIO_Init();
    SC_TIMER_Init();
    TouchKeyInit();
    /*write initial function here*/

    EA = 1;
}

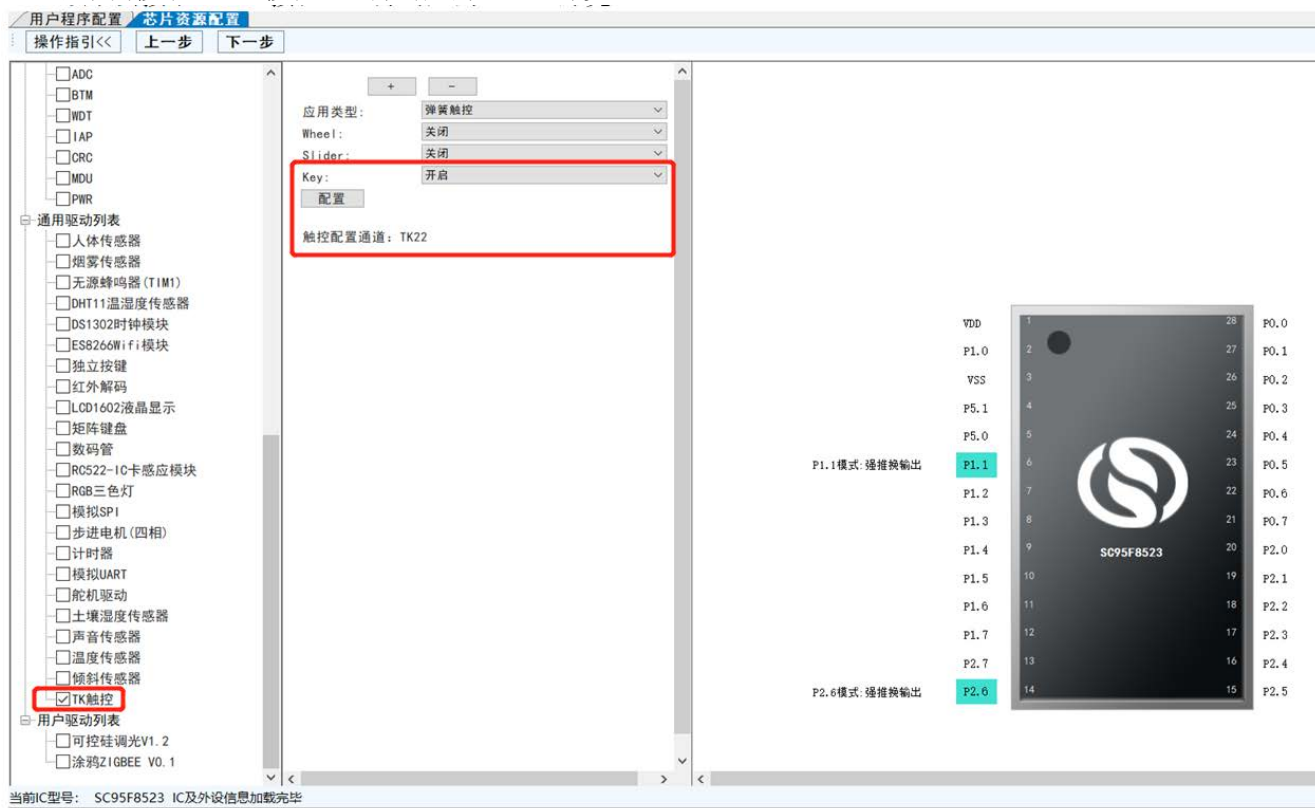
/*****
*函数名称: SC_OPTION_Init
*函数功能: OPTION配置初始化函数
*入口参数: void
*出口参数: void
******/
void SC_OPTION_Init(void)
{
    /*OPTION_Init write here*/
}

/*****
*函数名称: SC_GPIO_Init
*函数功能: GPIO初始化函数
*入口参数: void
*出口参数: void
******/
void SC_GPIO_Init(void)
{
    GPIO_Init(GPIO0, GPIO_PIN_3,GPIO_MODE_OUT_PP);
    GPIO_Init(GPIO0, GPIO_PIN_6,GPIO_MODE_OUT_PP);
    GPIO_Init(GPIO5, GPIO_PIN_2,GPIO_MODE_OUT_PP);
    GPIO_Init(GPIO5, GPIO_PIN_4,GPIO_MODE_OUT_PP);
    GPIO_Init(GPIO4, GPIO_PIN_7,GPIO_MODE_OUT_PP);
    /*GPIO_Init write here*/
}

void Sys_Scan(void)
{
    if(TKScanflag == 1)
    {
        if(SOCAPI_TouchKeyStatus&0x80) //重要步骤2: 触摸键扫描一轮标志, 是否调用TouchKeyScan()一定要根据此标志位置起后
        {
            SOCAPI_TouchKeyStatus &=0x7f; //重要步骤3: 清除标志位, 需要外部清除。
            exKeyValueFlag = TouchKeyScan();
            ChangeTouchKeyvalue(); //键值转换
            TKScanflag=0;
            return ;
        }
    }
    if(TKScanflag==0)
    {
        LED();
        SOCAPI_TouchKeyStatus &=0x7f; //重要步骤3: 清除标志位, 需要外部清除。
        TouchKeyRestart(); //启动下一轮转换
        TKScanflag=1;
    }
}
```

4.17.2 SDK1213 TK 示例

该示例按下 TK24 按键，上方对应的 LED1 灯亮



在易码魔盒通用外设驱动中勾选 TK 触控设置应用类型（该示例值用了 Key 所以只有 Key 选择开启），在生成代码时软件会自动将对应的 TK 库加入工程中，设置好后点击配置调试 TK 按键参数生成配置信息文件。

```
void main(void)
{
    /*<UserCodeStart>*/
    uint32_t KeyValue;
    /*<UserCodeEnd>*/
    /*** MCU初始化函数 ***/
    SC_Init();
    /*<UserCodeStart>*/  
/*<SinOne-Tag><34>
    GPIO_WriteHigh(GPIO1,GPIO_PIN_0);

    /*<UserCodeEnd>*/  
/*<SinOne-Tag><34>
    /*<UserCodeStart>*/  
/*<SinOne-Tag><26>
    TouchKeyInit();
    /*<UserCodeEnd>*/  
/*<SinOne-Tag><26>
    /*<UserCodeStart>*/  
/*<SinOne-Tag><25>
    while(1)
    {
        /*<UserCodeStart>*/  
/*<SinOne-Tag><28>
        if(SOCAPI_TouchKeyStatus & 0X80)
        {
            /*<UserCodeStart>*/  
/*<SinOne-Tag><30>
            SOCAPI_TouchKeyStatus &= 0x7f;
            /*<UserCodeEnd>*/  
/*<SinOne-Tag><30>
            /*<UserCodeStart>*/  
/*<SinOne-Tag><29>
            KeyValue = TouchKeyScan();
            /*<UserCodeEnd>*/  
/*<SinOne-Tag><29>
            /*<UserCodeStart>*/  
/*<SinOne-Tag><33>
            if(KeyValue & (0x011 << 22))
            {
                GPIO_WriteHigh(GPIO2,GPIO_PIN_6);
            }
            else
            {
                GPIO_WriteLow(GPIO2,GPIO_PIN_6);
            }
            /*<UserCodeEnd>*/  
/*<SinOne-Tag><33>
            /*<UserCodeStart>*/  
/*<SinOne-Tag><32>
            TouchKeyRestart();
            /*<UserCodeEnd>*/  
/*<SinOne-Tag><32>
        }
        /*<UserCodeEnd>*/  
/*<SinOne-Tag><28>
    }
    /*<UserCodeEnd>*/  
/*<SinOne-Tag><25>
}
```

在程序主循环中 if(SOCAPI_TouchKeyStatus)判断 TK 扫描是否完成，扫描完成后通过 TouchKeyScan 获取键值然后进行处理，处理完毕后再 TouchKeyRestart 重新开启 TK 扫描。

4.18 功能复用示例

4.18.1 SDK1211/1212 功能复用示例

功能复用示例是对学习评估板各种资源整体上的一个运用。示例通过 TK 按键与开关按键选择演示学习评估板不同的功能模块。TK 按键可选择进入不同的模式，开关按键为确认键与退出键。

1. SC95F8617/SC95F8616 按键说明：

开关按键 S1：演示功能确认键

开关按键 S2：退出当前演示的功能

TK24：代表二进制 0

TK25：代表二进制 1

TK26：重新输入

若选择模式 3，则需输入二进制 0011，即按下两下 TK24，两下 TK25（此过程中按下 TK26 可重新选择模式）。再按确认键即可看到模式 3 所演示功能，按退出键则返回模式选择界面。

2. SC95F8617/SC95F8616 按键说明：

开关按键 S1：模式选择，每按一次模式加一

开关按键 S2：演示功能确认键或退出当前演示功能

若选择模式 3，则需通过 S1 按键选至模式 3。再按 S2 按键即可看到模式 3 所演示功能，再按 S2 则退出，返回模式选择界面。

3. 模式说明：

模式 1：PWM 控制三色灯，现象为三色灯循环变色。

模式 2：Timer1 计数，若按下 S1 键两次则 LED1 亮灭状态改变。

模式 3：UART1 通过 TXD1 口往串口助手上位机发送数据，上位机显示字符串“TEST”，波特率为 9600

模式 4：ADC 测温，显示当前温度。

模式 5：SPI 操作 W25Q16，将 10 个数写进 W25Q16，通过 TXD0 口发送数据到串口，观察所读的值是否为所写的值。需将 J1 与 J2 用跳线帽短接。

模式 6：T2 捕获模式，将 SCK 脚与 T2EX 脚用杜邦线短接，LED 上显示 SCK 脚方波的周期。

模式 7：IAP 操作 ROM，将 10 个数写进 ROM，通过 TXD1 口发送数据到串口，观察所读的值是否为所写的值。

```
void main(void)
{
    SC_Init();
    while(1)
    {
        Sys_Scan(); //TK按键扫描
        if(TOFlag10ms)
        {
            TOFlag10ms = 0;
            Lcd_Display();
        }
        if(GetS1State() != RESET)
        {
            Mode = LcdDataTab[3] + LcdDataTab[2]*2 + LcdDataTab[1]*4 + LcdDataTab[0]*8; //模式选择

            switch(Mode)
            {
                case 1: Mode1_PWM(); break;
                case 2: Mode2_Timer1Counter(); break;
                case 3: Mode3_UART1(); break;
                case 4: Mode4_ADC(); break;
                case 5: Mode5_SPI(); break;
                case 6: Mode6_Timer2Capture(); break;
                case 7: Mode7_IAP(); break;
                default: break;
            }
        }
    }
}
```


4.18.1 SDK1213 功能复用示例

功能复用示例是对学习评估板各种资源整体上的一个运用。示例通过按键选择演示学习评估板不同的功能模块。

1. SC95F8523 按键说明:

TK24: 进入/退出演示功能。

S1: 功能选择。(注意 T2 和 S2 按键共用引脚)

2. SC95F7523 按键说明:

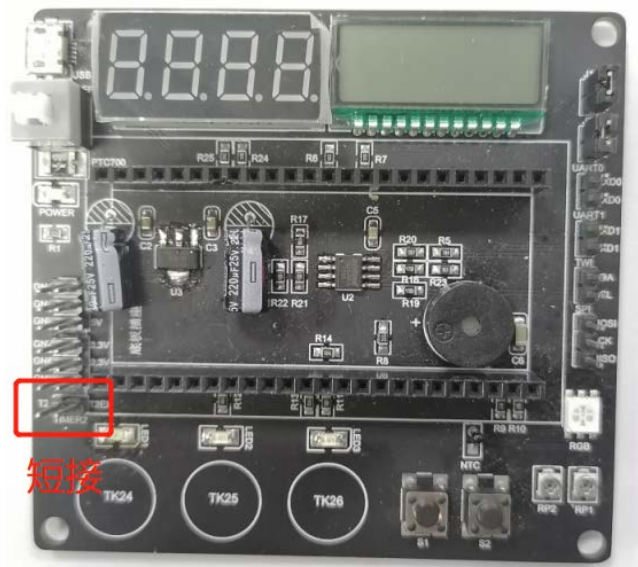
S1: 单机功能选择, 长按进入/退出演示功能。(注意 T2 和 S2 按键共用引脚)

将多数软件资源整合, 用户可通过按键选择所要观察的现象。

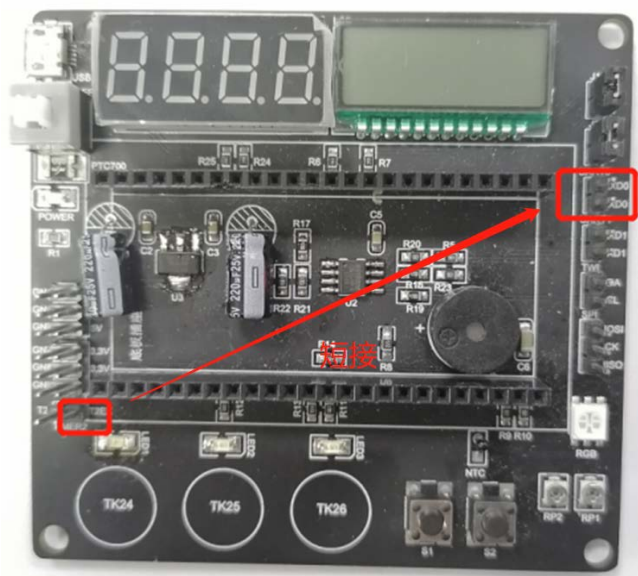
3. 模式说明:

Mode0: ADC 采集 NTC 热敏电阻电压, 转换为当前室温, 数码管显示室温。

Mode1: SC95F8523: 将 Timer2 设为捕获模式, Timer4 控制 P0.7 输出周期 20ms 的方波, 将 Timer2 短接, 数码管上显示该方波的周期, 单位毫秒。

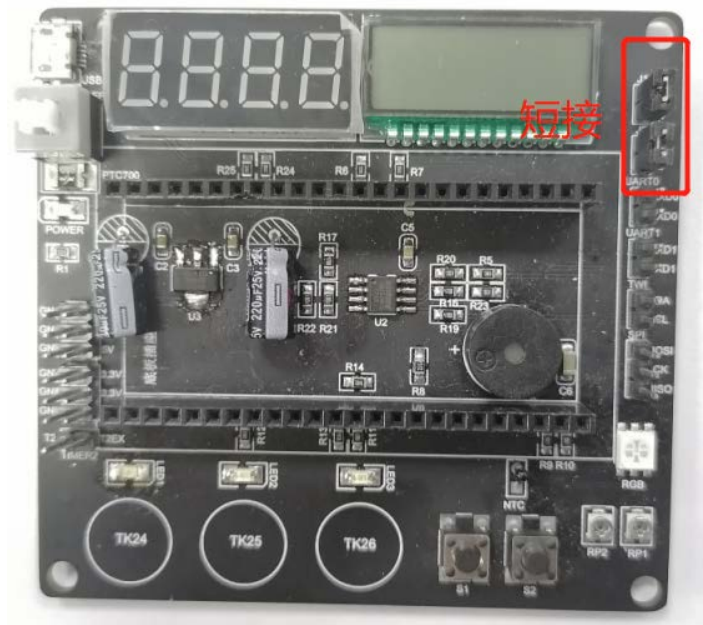


SC95F7523: 将 Timer2 设为捕获模式, Timer4 控制 P1.2/P1.3 输出周期 20ms 的方波, 将 Timer2 T2EX 短接, 数码管上显示该方波的周期, 单位毫秒。



Mode2: UART0 与上位机通信，波特率 9600，上位机发送数据到 MCU，MCU 将所接收到的内容发出。

Mode3: 通过 SPI 操作 W25Q16，将数组数据 0~9 写入 W25Q16，再将 W25Q16 中的值读出来并通过 UART0 发送给上位机，上位机显示 0~9。



Mode4: 通过改变 PWM6 占空比，实现 LED1 呼吸灯效果。

4.19 规格更改记录

版本	记录	日期
V1.0	初版	2019 年 12 月
V1.1	加入 SDK1213	2021 年 3 月